UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CONTRIBUTIONS TO THE JOINT CLASSIFICATION AND SEGMENTATION OF SEQUENCES

(My two cents on decoding and handwriting recognition)

dissertation presented to apply
for the Doctor of Philosophy degree

By: Salvador España Boquera
Directed by: Dr. María José Castro Bleda
Universitat Politècnica de València

Submitted on September 2015
Defended on January 2016

To my family, past, present and future,
very specially to my parents
and to my wife Manuela
for their unconditional love and support.


HI! SALVA PHD IS FINALLY RELEASED!
COME ON! LET'S GONNA READ IT!
SO... WHAT?
MMM... I'LL BETTER WAIT 'TIL THE MOVIE COMES OUT!

ABSTRACT

This work is focused on problems satisfying two properties: 1) they can be represented (at least approximately) in terms of one-dimensional sequences, and 2) solving these problems entails breaking the observed sequence down into segments which are associated to units taken from a finite repertoire. The segmentation and classification tasks usually required to determine the sequences of hidden units which best explain the observed signal are so intrinsically interrelated (fact known as "Sayre's Paradox") that they have to be performed jointly. Well known examples of these problems include automatic speech recognition (ASR) and handwritten text recognition (HTR).

In order to propose some contributions to this mature field of research, we have been inspired by what some works call the "successful trilogy", which refers to the synergistic improvements obtained when the following three points are considered:

1. a good formalization framework and powerful algorithms;
2. a clever design and implementation taking the best profit of hardware;
3. an adequate preprocessing and a careful tuning of all heuristics.

We describe and study "two stage generative models" (TSGMs) comprising two stacked probabilistic generative stages without reordering. This model not only includes the Hidden Markov Models (HMMs), in widespread use, but also the more general "segmental models" (SMs).

The "two stage decoder" may be deduced by simply running a TSGM in reversed way and introducing non determinism when required: A directed acyclic graph (DAG) is generated in a first stage and is used, together with a language model (LM), to determine the best sequences of hidden units explaining the observed signal. The well known one-pass decoder (better suited for the common case of combining HMMs with regular LMs such as n-grams) is a particular case and a thorough comparison between them and with alternative approaches is provided.

A formalization of parsing and decoding in terms of semiring values and language equations proposes the use of recurrent transition networks (RTNs) as a normal form for Context Free Grammars (CFGs), using them in a parsing-as-composition/intersection paradigm where parsing/decoding of context free languages is a slight extension of regular ones. This also leads to the proposal of a novel transducer composition algorithm that can work with RTNs and can deal with null transitions without resorting to filter-composition even in the presence of null transitions and using non-idempotent semirings.

Relating LMs, a review and some contributions mainly focused on LM interfaces, LM representation and on the evaluation of Neural Network LMs (NNLMs) are provided.

A review of SMs includes the combination of generative and discriminative segmental models and general scheme of frame emission and another one of SMs.

Some fast specialized Viterbi lexicon decoders taking profit of particular HMM topologies are proposed. They are able to manage sets of active states without requiring any kind of dictionary look-up (e.g. hashing) and are also cache-friendly.

A dataflow architecture allowing the design of flexible and diverse recognition systems from a little repertoire of components has been designed and implemented. The components can be expressed in a reactive way and a novel DAG serialization protocol allows DAG generators to emit a DAG while it is being generated and DAG decoders to process it while it is being received.

DAG generators can take over-segmentation constraints into account, make use SMs other than HMMs, take profit of the specialized decoders proposed in this work and use a transducer model to control its behavior making it possible, for instance, to use context dependent units.

Relating DAG decoders, they take profit of a general LM interface that can be extended to deal with RTNs.

Another widely used type of decoder, known as "one pass" is very suitable when we are limited to HMMs and finite state LMs (such as n-grams). Some improvements are proposed for this type of decoder by combining the specialized lexicon decoders and the "bunch" extension of the LM interface so that the complete decoder may avoid any kind of explicit dictionary look-up while allowing for an adequate parallelization.

The experimental part is mainly focused on HTR tasks on different input modalities (offline, bimodal). We have proposed some novel preprocessing techniques for offline HTR which replace classical geometrical heuristics and make use of automatic learning techniques (neural networks). Experiments conducted on the IAM database using this new preprocessing and HMM hybridized with Multilayer Perceptrons (MLPs) have obtained some of the best results reported for this reference database. Among other HTR experiments described in this work, we have used over-segmentation information, tried lexicon free approaches, performed bimodal experiments and experimented with the combination of hybrid HMMs with holistic classifiers.

RESUMEN

Este trabajo se centra en problemas que cumplen dos propiedades: En primer lugar, problemas que pueden representarse (al menos aproximadamente) en términos de secuencias unidimensionales. En segundo lugar, la resolución de estos problemas implica la descomposición de la secuencia observada en segmentos que se pueden clasificar en un conjunto finito de unidades. Las tareas de segmentación y de clasificación necesarias para obtener las secuencias de unidades que mejor explican la señal observada están tan intrínsecamente interrelacionadas (hecho conocido como "La paradoja de Sayre") que deben realizarse de manera conjunta. El reconocimiento automático del habla (ASR) y de la escritura manuscrita (HTR) son ejemplos de este tipo de tareas.

Para poder realizar aportaciones en un campo de investigación tan maduro como éste nos hemos inspirado por lo que algunos autores denominan "La trilogía exitosa", término que expresa la sinergia obtenida cuando se tienen en cuenta estos tres puntos:

1. un buen formalismo, que dé lugar a buenos algoritmos;
2. un diseño e implementación ingeniosos y eficientes, que saquen provecho de las características del hardware;
3. no descuidar el "saber hacer" de la tarea, un buen preproceso y el ajuste adecuado de los diversos parámetros.

Describimos y estudiamos "modelos generativos en dos etapas" sin reordenamientos (TSGMs). Estos modelos incluyen no sólo los ampliamente utilizados modelos ocultos de Markov (HMM), sino también los modelos segmentales (SMs).

Es fácil obtener un decodificador de "dos pasos" simplemente considerando a la inversa un TSGM introduciendo no determinismo cuando sea necesario: en primer lugar, se genera un grafo acíclico dirigido (DAG) que es utilizado, en un segundo paso, y conjuntamente con un modelo de lenguaje (LM), para obtener las secuencias de unidades que mejor explican la señal observada. El ampliamente conocido decodificador de "un paso" (adecuado para el caso habitual en que se combinan HMMs con LMs regulares tales como n-gramas) es un caso particular, con lo que se presenta una comparativa entre ambos decodificadores, así como con aproximaciones alternativas.

Se describe una formalización del proceso de decodificación basada en ecuaciones de lenguajes y semianillos. En ella se propone el uso de redes de transición recurrente (RTNs) como forma normal de gramáticas de contexto libre (CFGs) y se utiliza el paradigma de análisis por medio de composición/intersección de manera que el análisis de lenguajes incontextuales queda como una ligera extensión del análisis de lenguajes regulares. Este análisis también nos ha llevado a un algoritmo de composición de transductores que permite el uso de RTNs y que no necesita recurrir al concepto de composición de filtros incluso en presencia de transiciones nulas y con semianillos no idempotentes.

En relación a los LMs, se propone una extensa revisión y algunas contribuciones menores mayormente relacionadas con su interfaz, con la representación de algunos LMs y con la evaluación de LMs basados en redes neuronales (NNLMs).

También se ha realizado una revisión de SMs que incluye SMs basados en la combinación de modelos generativos y discriminativos, así como un esquema general de tipos de emisión de tramas y de SMs.

Se proponen varias versiones especializadas del algoritmo de Viterbi para modelos de léxico que sacan provecho de determinadas topologías de los HMMs. Estos algoritmos permiten estados activos sin, por tanto, tener que recurrir a estructuras de datos de tipo diccionario (como tablas de dispersión) y son capaces de sacar provecho de la *caché* por el tipo de acceso a los datos.

Se ha diseñado e implementado una arquitectura de flujo de datos o "*dataflow*" para obtener de modo muy flexible diversos tipos de reconocedor a partir de un pequeño conjunto de piezas básicas. Sus componentes se pueden describir de un modo "reactivo" y un novedoso protocolo de serialización de DAGs permite crear generadores de DAGs capaces de emitirlos a medida que se generan, así como decodificadores que pueden procesarlos mientras se van recibiendo.

Describimos generadores de DAGs que pueden tener en cuenta restricciones sobre la segmentación, utilizar modelos segmentales no limitados a HMMs, hacer uso de los decodificadores especializados propuestos en este trabajo y utilizar un transductor de control que permite el uso de unidades dependientes del contexto.

Los decodificadores de DAGs hacen uso de un interfaz bastante general de LMs que ha sido extendido para permitir el uso de RTNs.

Otro tipo de decodificador ampliamente utilizado es el conocido como "un paso", que resulta muy adecuado cuando nos limitamos a HMMs y a LMs de estados finitos (como los n-gramas). Se proponen también mejoras para este tipo de decodificador combinando las características de los algoritmos especializados para léxicos y la interfaz de LM en modo "*bunch*" de modo que obtenemos un decodificador de tipo un paso que evita cualquier tipo de búsqueda explícita de tipo diccionario ("*hashing*" o similar) y que permite conseguir una buena paralelización.

La parte experimental de este trabajo está muy centrada en reconocimiento de escritura en diversas modalidades de adquisición (*offline, bimodal*). Hemos propuesto técnicas novedosas para el preproceso de escritura manuscrita *offline* que evita el uso de heurísticos geométricos prefiriendo utilizar, en su lugar, técnicas de aprendizaje automático (redes neuronales). Con este preproceso y utilizando HMMs hibridados con redes neuronales hemos conseguido, para la base de datos IAM, algunos de los mejores resultados publicados. Entre los experimentos relacionados con este trabajo podemos mencionar el uso de información de sobre-segmentación, aproximaciones sin restricción de un léxico, experimentos con datos bimodales o la combinación de HMMs híbridos con reconocedores de tipo holístico.

RESUM

Aquest treball es centra en la resolució de problemes on es compleixen dues propietats: en primer lloc, problemes on les dades es poden representar (almenys aproximadament) mitjançant seqüències unidimensionals. En segon lloc, cal trencar la seqüència observada en segments que poden pertanyer a un nombre finit de tipus. Sovint, ambdues tasques es relacionen de manera tan estreta que resulta impossible separar-les (fet conegut com "paradoxa de Sayre") i s'han de realitzar de manera conjunta. Entre els diversos exemples d'aquest tipus de problema podem mencionar el reconeiximent automàtic de la parla (ASR) i de l'escriptura manuscrita (HTR).

Per tal de poder realitzar aportacions en un camp d'investigació tan consolidat com aquest, ens hem inspirat pel que alguns autors anomenen la "trilogia exitosa", terme que representa la millora que es pot aconseguir de manera sinèrgica quan prenim en compte:

1. un bon formalisme, que done lloc a bons algorismes;
2. un diseny i una implementació eficients, amb ingeni, que facen bon us de les particularitats del maquinari;
3. no perdre de vista el "saber fer", emprar un preprocés adequat i fer bon us dels diversos paràmetres.

Descrivim i estudiem "models generatiu amb dues etapes" sense reordenaments (TSGMs). Aquestos models no sols inclouen els àmpliament coneguts models ocults de Markov (HMM), sinò també els models segmentals (SM).

És fàcil obtindre un decodificador "en dues etapes" simplement considerant a l'inrevés un TSGM introduint no determinisme quan siga necessàri: en primer lloc, es genera un graf acíclic dirigit (DAG) que és emprat, en un segon pas, i conjuntament amb un model de llenguatge (LM), per obtindre les seqüències d'unitats que millor expliquen la senyal observada. L'àmpliament conegut decodificador "d'un pas" (adequat per al cas freqüent on es combinen HMMs amb LMs regulars com els n-grames) és un cas particular, motiu pel qual es presenta una compartiva entre ambdós decodificadors, així com amb aproximacions alternatives.

Es descriu una formalització del procés de decodificació basada en equacions de llenguatges i en semianells. En ella proposem emprar xarxes de transició recurrent (RTNs) com forma normal de representar gramàtiques incontextuals (CFGs) i s'empra el paradigma d'anàlisi sintàctic mitjançant composició/intersecció de manera que l'anàlisi de llenguatges incontextuals es pot expressar com una lleugera extensió de l'anàlisi de llenguatges regulars. Aquest anàlisi ens ha dut a proposar un algorisme de composició de transductors que permet emprar RTNs i que no necessita recorrer al concepte de composició amb filtres fins i tot en presència de transicions nul·les i amb semianells no idempotents.

En relació als LMs, es proposa una extensa revisió i algunes contribucions majorment relacionades amb la seva interfície, amb la representació d'alguns LMs i amb l'avaluació de LMs basats en xarxes neuronals (NNLMs).

També s'ha realitzat una revisió de SMs que inclou SMs basats en la combinació de models generatius i discriminatius, així com un esquema general de tipus d'emissió de trames i altre de SMs.

Es proposen diverses versions especialitzades de l'algorisme de Viterbi per a models de lèxic que trauen profit de determinades topologies dels HMMs. Aquestos algorismes permeten emprar estats actius sense, per tant, haver de recórrer a estructures de dades de tipus diccionari (com taules de dispersió) i, pel seu tipus d'accés a les dades, són capaços de traure profit de la *caché*.

S'ha disenyat i implementat una arquitectura de flux de dades o "*dataflow*" per obtindre de manera molt flexible diversos tipus de reconeixedor a partir d'un xicotet conjunt de peces bàsiques. Els seus components es poden descriure de manera "reactiva" i un novedós protocol de serialització de DAGs permet crear generadors de DAGs capaços d'emetre'ls mentre es generen, així com decodificadors que els poden processar mentre van sent rebuts.

Descrivim algorismes per generar DAGs capaços de tindre en compte restriccions sobre la segmentació, emprar models segmentals no limitats a HMMs, fer us dels decodificadors especialitzats proposats en aquest treball i emprar un transductor de control que permet, per exemple, emprar unitats dependents del contexte.

Els decodificadors de DAGs fan us d'una interfície de LMs prou general que ha segut extesa per permetre l'ús de RTNs.

Altre tipus de decodificador àmpliament utilitzat és el conegut com "un pas", que resulta molt adequat quan ens limitem a HMMs i a LMs d'estats finits (com els n-grames). Es proposen també millores per a aquest tipus de decodificador combinant les característiques dels algorismes especialitzats per a lèxics i la interfície de LMs en mode "*bunch*" de manera que obtenim un decodificador de tipus un pas que evita qualsevol tipus de cerca explícita de tipus diccionari (per exemple "*hashing*") i que permet aconseguir una bona paral·lelització.

La part experimental d'aquest treball està molt centrada en el reconeiximent d'escriptura en diverses modalitats d'adquisició (*offline*, *bimodal*). Proposem un preprocés d'escriptura manuscrita *offline* evitant l'us d'heurístics geomètrics, en el seu lloc emprem classificadors basats en tècniques d'aprenentatge automàtic (xarxes neuronals). Amb aquest preprocés i emprant HMM hibridats amb xarxes neuronals hem aconseguit, per a la base de dades IAM, alguns dels millors resultats publicats. Entre els experiments relacionats amb aquest treball també podem mencionar l'ús d'informació de sobre-segmentació, aproximacions sense restricció a un lèxic, experiments amb dades bimodals o la combinació de HMMs híbrids amb classificadors holístics.

AUTHOR'S PREFACE

This report, by its very length,
defends itself against the risk of being read.
Winston Churchill

Je n'ai fait celle-ci plus longue que parce que
je n'ai pas eu le loisir de la faire plus courte[1]
Blaise Pascal, Provincial Letters: Letter XVI

They muddy the water, to make it seem deep
Friedrich Nietzsche, Thus Spoke Zarathustra

There is nothing new under the sun
Ecclesiastes, 1:9

A stupid man's report of what a clever man says
can never be accurate, because he unconsciously translates
what he hears into something he can understand.
Bertrand Russell

Quoting Pascal, I would like to present my apologies to the reader for not devoting time enough to write a shorter document instead of what, at a first glance, seems a denial-of-service attack.[2] Please, do not expect this length to be proportional or due to the number of novelties. Do not expect a radically new approach, either. I have read a lot of PhD reports during the time I have devoted to accomplish this work and I have observed that many of them claimed new algorithms, approaches, unified frameworks,... *new whatever*. I felt disappointed[3] since it seemed to me that some of them were just reinventing the wheel or adapting old ideas.

Most ideas you will find here are just common sense: The same wheel everybody "who is friend with his/her brain and who knows that 2+2=4" could probably reinvent *when faced to the same problems under the same conditions*. That is probably why I have devoted so much time (reflected in space) to review and to organize previous art, trying to convince me that there was nothing new under the sun, trying to show I was reinventing the wheel before claiming mistaken attributions. We have to be very cautious when using words like "new", "general" or "unifying" because we cannot assure, no matter how much time we devote to literature reviews, that a work is free from reinvented wheels. I also apologize for the moments when enthusiasm has made me forgot that.

1 "I would have written a shorter letter, but I did not have the time."
2 It is *not* the case although, perhaps, it is a pre-emptive defense, as in Churchill's quote.
3 Reciprocally, I will surely disappoint.

I advise now (too late) against devoting too much effort to reviewing previous art. Such a high effort does not pay out for several reasons:

1) Making a review means that there is not a set of bibliographic references that *perfectly fits* what we wanted to tell. Does it implicitly denigrate previous reviews and books on the field? I hope not!

2) The review effort is not to be appreciated since it basically consists in writing things too trivial, boring and irritating for readers already introduced in the field. Indeed, many ideas are just, for clever people, too trivial to detail, or they are described loosely for laziness or for lack of space (specially in proceedings). Besides, implementation details are often considered *second rate* research, usually left for scholarship holders fated to rediscover them. In other cases, they may be hidden as "industrial secrets". Many ideas are obvious only a posteriori (it may be related to "one way functions" used in cryptography).

3) I have tried to explain things in the most plain way, trying to reduce the use of definitions and mathematical formulas.[4] The additional effort required to simplify is not only not to be appreciated but, quoting Nietzsche, is advised against your own interest of making your own work to seem deep.

4) Finally, the natural procedure to better explain things is to place your contributions where it is more logically suited (like "seasoning" with a little pinch[5] of contributions a huge "meal"). This is different from the classical PhD structure (more practical, with higher "expected return") consisting in briefly referring the user to a list of references in order to present the contributions afterwards. I am aware that most readers, when facing such unconventional structure, would find it not only inconvenient but, alarmingly, would wrongly believe that contributions are restricted to the chapter of experimental results (e.g. comparing the WER with other ones) hence bypassing the rest. Ironically, I am much more proud of the results of the first parts where novel algorithms appear that have not yet an experimental counterpart.

It is really easy to obtain lots of results by running scripts when entering a research group with an established infrastructure and scripts of prior experiments. We cannot say the same when tools have been developed by ourselves from scratch over several years and when we have changed to a novel domain, things are always more difficult to assess than expected. When I began this PhD, some years ago, I was *persuaded* to work on spoken dialog tasks, but I felt frustrated by the lack of flexibility of toolkits and decoders we had at our disposal. I was rather interested in the "well solved" decoding process. I finally decided to start a new toolkit where Francisco Zamora and Jorge Gorbe contributed so much.[6] After all this time, this PhD report is more the personal notes I would have liked to have had at the beginning rather than the work I have to deliver now. For this reason, I hope that someone new to the field might find here something useful for them.

4 After studying Computer Science, before this PhD, I studied a math degree (best student award) and I am somewhat self convinced against abusing of formulas.

5 Hence the subtitle *"My two cents on..."*

6 Lately, Adrián Palacios and Joan Pastor have also made contributions. Also, Pako Zamora, who made a great PhD work including their own features in April, has been made an astonishing work developing a fork of this toolkit, April-ANN, which mainly corresponds to the neural networks part. At the moment of this writing, decoders remain out of this fork, but we expect to be able to include them when possible.

ACKNOWLEDGMENTS

This thesis would not have been possible without the help of so many people that I will surely miss many of them. First of all, I want to acknowledge my family for their infinite patience and unconditional support.

I should like to thank my advisor, Dr. María José Castro, for guiding me towards the use of hybrid HMMs and neural network language models. I have ignored many of her continous advices which would probably make me finish a PhD work much before and with less effort, so I am enterly responsible of the lot of delays and mistakes which can be undoubtedly found here.

I have had the pleasure and the honour to work with Francisco (Pako) Zamora and with Jorge Gorbe during many years, mainly developing the April toolkit. We have spent uncountable afternoons brainstorming or even debugging in face of a screen projector. The work presented here would not be the same without their help. The work of Pako on the toolkit can only be described as atonishing. More recently, but not less important, I had also the pleasure to work with Joan Pastor and, during some months, with Adrián Palacios. Joan is using April to extend many ideas on handwriting recognition and to propose new ones for his own PhD, and it is also a pleasure to collaborate with him.

I want to thank those colleages from UPV and UJI universities that have given me advices and support. Among them, David Guerrero, Fernando Alvarruiz and Jose Román. I also want to thank Moisés Pastor for lending us some of the handwritten recognition models of his PhD in order to been able to compare with them.

I had the pleasure to be invited for some months at "Laboratoire Informatique d'Avignon" (LIA) in 2010, where I was very wellcome. This is a really nice and familiar place to work and I want to thank all of them. I believe that I learned many interesting things there. I want to thank (in alphabetical order): Nathalie Camelin, Renato De Mori, Fred Duvert, Marc El Bèze, Juliette Kahn, Georges Linarès, Driss Matrouf, Pascal Nocéra Michael Ruvier, Greg Senay, and many others. I also want to thank to Jeannot and Mireille Arakelian, and to Teresa Mesas, Patrick and Damien Lecoq for being so kind with me in Avignon, they were a second family.

I would also like to thank Prof. Dr. Horst Bunke and other members of the research group from IAM at Bern University. As with people from LIA, we can observe that behind the high quality of their work there are also extraordinary human qualities.

I'm very grateful to the reviewers and to anyone who takes the effort to read even little parts of this painful long work or who has contributed to help to improve it with suggestions.

I have the opportunity to ask several questions related to typography and to the bookbinding of this report to Maite Giménez, I'm very grateful for her kindness and for her advices.

Also, since reality is probably much more complex that we can even grasp, I would like to devote part of my acknowledgements to whatever or whoever that wherever or whenever has provided some help even if I have not been remotely aware of it.

Finally, I want to thank all the authors I have cited in these pages (even those I have mainly cited to criticize) since I have learned a lot from them and this PhD can only be understood as a tribute to their work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

NOMENCLATURE

$p(\cdot)$	Probability distribution
$p(\cdot \mid \cdot)$	Conditional distribution
D	Set of samples
X	Space of features
Ω	Lexicon (set of lexical units)
A	Set of possible Actions
θ	Model parameters
Θ	Parameter space
Λ	Model parameters
λ	loss function
R	Overall risk
S	State space (or state of nature)
$\alpha(x)$	decision
$R(\alpha \mid x)$	Conditional risk
$A \perp\!\!\!\perp B$	Independence of random variables "A is independent of B"
$A \perp\!\!\!\perp B \mid C$	Conditional independence of random variables "A is independent of B given C"
$[q, q']$	Shortest distance between FSA/RTN states q and q'
$p_e(q, q')$	Set of paths from q to q' represented by sequences of *edges*
$p_{st}(q, q')$	Set of paths from q to q' defined as *state* sequences
$\langle q, r \rangle$	denotes $[q, q_F] \circ [r, r_F]$
$\langle q, q', r \rangle$	denotes $[q, q'] \circ [r, r_F]$

ACRONYMS

ANN	Artificial Neural Network
API	Application Programming Interface
ATN	Augmented Transition Network
ASR	Automatic Speech Recognition
BLEU	BiLingual Evaluation Understudy
BFS	Breadth First Search
BLSTM	Bidirectional Long Short-Term Memory
BOS	Begin Of Sentence (context cue)
BN	Bayesian Network
BRNN	Bidirectional Recurrent Neural Network
BW	Baum Welch
CC	Cepstral Coefficients / Connected Component
CI	Conditional Independence
CER	Character Error Rate / Concept Error Rate
CF	Context Free
CFG	Context Free Grammar
CML	Conditional Maximum Likelihood
CNF	Chomsky Normal Form
CNN	Convolutional Neural Network
CoAP	Co-ocurrence Agreement Probability
CPU	Central Processing Unit
CRF	Conditional Random Field
CTC	Connectionist Temporal Classification
CYK	Cocke Younger Kasami
DAG	Directed Acyclic Graph
DAWG	Directed Acyclic Word Graph
DBN	Dynamic Bayesian Network
DCT	Discrete Cosine Transform

DFS	Depth First Search
DGM	Dynamic Graphical Model
DP	Dynamic Programming
dpi	Dots Per Inch
DSP	Digital Signal Processor
DTW	Dynamic Time Warping
EBM	Energy Based Model
EBNF	Extended Backus Naur Form
EBW	Extended Baum Welch
ECFG	Extended Context Free Grammar
EM	Expectation Maximization
EOS	End Of Sentence (context cue)
ERM	Empirical Risk Minimization
FB-NNLM	Fall Back Neural Network Language Model
FBS-NNLM	Fall Back Skipping Neural Network Language Model
FER	Frame Error Rate
FF	Frequency Filtering
FFT	Fast Fourier Transform
FG	Factor Graph
FIFO	First In First Out
FOM	Figure Of Merit
FSA	Finite State Automata
FSM	Finite State Model
GLC	Generalized LC
GLL	Generalized LR
GLR	Generalized LL
GM	Graphical Model
GMM	Gaussian Mixture Model
GNF	Greibach Normal Form
GPU	Graphics Processing Unit
GSF	Grammar Scale Factor
GSS	Graph Structured Stack

GTN	Graph Transformer Network
KSPC	Key Strokes Per Character
KSR	Key Stroke Ratio
KWS	Keyword Spotting
HDM	Hidden Dynamic Model
HMM	Hidden Markov Model
HSMM	Hidden Semi-Markov Model
HTM	Hidden Trajectory Model
HTR	Handwritten Text Recognition
Hz	Hertz
IPR	Interactive Pattern Recognition
LALR	Look-Ahead LR
LC	Left-Corner
LCA	Lowest Common Ancestor
LDA	Linear Discriminative Analysis
LL	Left-to-right, Leftmost derivation
LMLA	Language Model Look-Ahead
LM	Language Model
LR	Left-to-right, reversed Rightmost derivation
LSA	Latent Semantical Analysis
LVCSR	Large Vocabulary Speech Recognition
LVQ	Learning Vector Quantization
MAP	Maximum A Posteriori (estimation/training)
MBR	Minimum Bayes Risk
MCE	Minimum Classification Error
ME	Maximum Entropy
MELM	Maximum Entropy Language Model
MEMM	Maximum Entropy Markov Model
MERT	Minimum Error Rate Training
MEU	Maximum Expected Utility
MFCC	Mel Frequency Cepstral Coefficients

MGGI	Morphic Generator Grammatical Inference (methodology)
ML	Maximum Likelihood / Machine Learning
MLE	Maximum Likelihood Estimation
MLP	Multilayer Perceptron
MMI	Maximum Mutual Information
MPE	Minimum Phone Error / Most Probable Explanation
MPF	Marginalize a Product Function
MRF	Markov Random Field
MRL	Meaning Representation Language
NE	Named Entity
NER	Named Entity Recognition
NFG	Normal Factor Graph
NLP	Natural Language Processing
NLU	Natural Language Understanding
NNLM	Neural Network Language Model
OCR	Optical Character Recognition
OOS	Out Of Shortlist
OOV	Out Of Vocabulary
PCA	Principal Component Analysis
PCFG	Probabilistic Context Free Grammar
PDF	Probabilistic Distribution Function / Probabilistic Density Function
POS	Part Of Speech
PPL	Perplexity
PPT	Pronunciation Prefix Tree
PR	Pattern Recognition
RBF	Radial Basis Function
RELU	Rectified Linear Unit (activation function)
REMAP	Recursive Estimation Maximization A Posteriori
RGR	Rich Get Richer (strategy)
RLSA	Run Length Smoothing Algorithm
RNN	Recurrent Neural Network

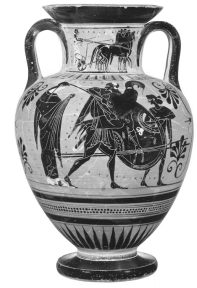
ROC	Receiver Operating Characteristic
RRPG	Regular Right Part Grammar
RTF	Real Time Factor
RTN	Recurrent Transition Network
SCR	Spoken Content Retrieval
SER	Sentence Error Rate
SIMD	Single Instruction Multiple Data
SLU	Spoken Language Understanding
SM	Segment/Segmental Model
SMT	Statistical Machine Translation
SPN	Sum Product Network
SSD	Solid State Drive
STD	Spoken Term Detection
STFT	Short Term Fourier Transform
SVG	Scalable Vector Graphics
TER	Translation Edit Rate
TRAP	TempoRAl Pattern
TSGM	Two Stage Generative Model
TTS	Text To Speech
UBM	Universal Background Model
UTD	Unsupervised Term Discovery
VAD	Voice Activity Detection
VTLN	Vocal Tract Length Normalization
WB	Witten-Bell (discount)
WCFG	Weighted Context Free Grammar
WER	Word Error Rate
WIP	Word Insertion Penalty
WFST	Weighted Finite State Transducer / Well Formed Substring Table
WMT	Workshop of Machine Translation
WPM	Words Per Minute
WSR	Word Stroke Ratio

1 INTRODUCTION

INUENIOADMIRANSNUMERUMMATRESQUEUIROSQUE
COLLECTAMEXILIOPUBEMMISERABILEUOLGUS
UNDIQUECONUENEREANIMISOPIBUSQUEPARATI[1]

Virgil, Aeneid

1.1 MOTIVATION



The fourth-century grammarian Servius criticized his colleague Donat for reading, in Virgil's Aeneid, the words *collectam ex Ilio pubem* ("a people gathered from troy") instead of *collectam exilio pubem* ("a people gathered for exile"). Such interpretation mistakes were common when reading ancient scrolls which neither separated words nor made a distinction between lower-case and upper-case letters, nor used punctuation. Before the IX Century, writing was much more related to speech: readers played aloud in order to allow the ear to disentangle what to the eye seemed a continuous string of signs [Manguel 1998; Saenger 2000].[2]

Despite the use of spaces and punctuation signs to solve segmentation problems in modern western scripts, other current script systems such as Chinese, Japanese or Thai do not provide delimiters to our notion of "word". Indeed, some studies [Sproat *et al.* 1996] remark an important rate of disagreement on segmentation ambiguities even among native speakers.

naive greedy approaches

It is possible to automatically detect all possible segmentation ambiguities when a lexicon is provided, and there are two really simple and naive techniques to remove them: On one side, the maximum matching greedy approach takes the longest word. This approach would lead, for the introductory example, to the solution proposed by Servius. On the other side, minimum matching chooses the shortest alternative. Despite their simplicity, the rationale behind these greedy approaches is not related to the task at hand and, indeed, they may be unable to find a solution in some cases, even when such a solution exists. Maybe, the problem between Servius and Donat does not have a final definitive answer so, perhaps, the best thing which can be done is to annotate both alternatives together with the reasons and consequences of each interpretation.

1 ...inuenio admirans numerum, matresque uirosque, collectam exilio pubem, miserabile uolgus. Undique conuenere, animis opibusque parati...

2 Many works such as [Gibrat and Dehaene 2012] mentions St. Ambroise as the first evidence of silent reading in antiquity (see `http://en.wikipedia.org/wiki/Ambrose#Ambrose_and_reading`) but, according to [Fenton 2006], this statement is a myth and silent reading was normally practiced in antiquity.

The absence of word segmentation ambiguities can be assured when the lexicon satisfies the *prefix condition*, meaning that no word is the prefix of another one.[3] Unfortunately, it is not possible to choose or to design the lexicon in most real word problems (it is rather given as part of the problem specification) but, even if we could, this only would help to solve the problem in some particular cases. A text document without segmentation ambiguities may contain segmentation and classification ambiguities after being handwritten. Although handwritten lines are two-dimensional images, we can consider them as "fundamentally one-dimensional" if we chop them into small pieces in the order of the lecture (see Figure 1) and are able to represent the variability of image columns by means of fixed size feature vectors.

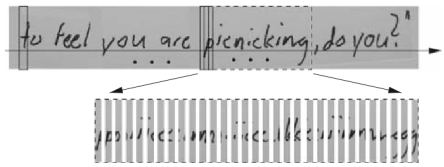


Figure 1: A handwritten line can be considered as a one-dimensional sequence when chopped into small pieces in the order of the lecture (taken from [Plötz and Fink 2009]).

One of the most interesting aspects of handwriting text recognition (HTR) is the interrelation between classification and segmentation. Figure 2 shows a handwritten line image where the same image segment is classified in several ways depending on the context where it appears.

Figure 2: The same image can be interpreted in different ways depending on the context where it appears. In this example, "fro**m**", "**cu**stomize" and "**in**trigue" share the same image segment which is associated different interpretations. This ambiguity can also appear at higher levels (for instance, a sole image can correspond to different words).

This relationship between classification and segmentation is known as the Sayre's paradox [Sayre 1973] page 217 Subsection 3.2.:

> *Identification of letters within a cursive line requires locating the beginning and the end points of individual letter-inscriptions. It is common to think of this as a task to be accomplished before the individual inscriptions can be recognized. But this is paradoxical, since the individual letters must be recognized as such before it can be determined where one inscription leaves off and another begins.*

Other authors drawn the same conclusion [Kaltenmeier *et al.* 1993]:

> *In cursive script the only apriori segmentable entities are whole words and not single characters. Hence, finding the right segmentation of a word is strongly related to the intrinsic classification*

3 A code satisfying this property is named *"instantaneous"*. It is also possible to demonstrate that some non instantaneous codes do not produce ambiguities when some word sequence constraints are also taken into account [Stockmeyer and Modha 2001].

> *task to be solved. Traditional "segmentation first" approaches might hardly be used for cursive script recognition. Some experiments were made with different approaches; none of them was really successful enough. Often, even for human readers, character segmentation can only be derived together with recognizing what was written, i.e. segmentation is performed implicitly during recognition. Technical systems should follow the same approach.*

The same problem has been observed in other tasks. For instance, in automatic speech recognition (ASR), [Konig *et al.* 1996] says:

> *A general formulation of statistical ASR can be summarized simply by a question: how can an input sequence (e.g., a sequence of spectral vectors) be explained in terms of an output sequence (e.g., a sequence of phones or words) when the two sequences are not synchronous (since there are multiple acoustic vectors associated with each pronounced word or phone)?*

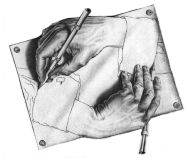
Also, in the field of image analysis, we can quote [Schwartzkopf 2002]:

> *Ji [Lundsteen and Piper 1989] recognized the dilemma that the classification needs correct information from segmentation, but that segmentation often needs correct information from classification as well. Adam and Dinstein [Agam and Dinstein 1997] also realized this and suggested combining the two steps for more accurate identification. Both [Agam and Dinstein 1997] and [Lundsteen and Piper 1989] recognized the potential usefulness of combining segmentation and classification, but provided no method to accomplish it.*

A more schematic way to formulate this "chicken-and-egg" problem is as follows [Vinciarelli and Lüttin 2000]:

> *A letter cannot be segmented before having been recognized and cannot be recognized before having been segmented.*

which can be further synthesized and paraphrased in a task independent way:



Classification requires segmentation.
Segmentation requires classification.

Before asking ourselves whether this paradox can be solved or not, should not we first try to determine if it can be avoided? Let us see two possible ways:

- in some cases, the entire sequence can be taken as a whole, as a sole thing to be classified. For instance, isolated word speech recognition can be solved at least in two ways: by constructing a model for every word or by considering words composed by sequences of sub-word models. In general, the curse of dimensionality limits the applicability of the holistic approach to tasks with very small lexicon sizes because the number of patterns required to train the models drastically grows with the number of model parameters;

- another way to avoid this paradox consists in asking the users for an explicit segmentation. For instance, before 1997, many commercially available large vocabulary speech recognition systems required the user to speak with pauses between words [Alleva *et al.* 1998]. Similarly, some paper forms contain boxes suited for writing isolated characters. This technique cannot be applied in all tasks, cannot completely solve the problem and is a nuisance to be avoided.

As we have seen, we cannot escape from solving the circular paradox. The way to solve it consists in breaking the circle by introducing non-determinism, i.e., by taking all the possible segmentation hypothesis into account and by choosing among them the best one by means of some objective function (Figure 3). Indeed, in order to classify something we need first to choose *what* is to be classified: it has to be broken down into parts or segmented. Thus, a search space is defined and the task of finding the best hypothesis is known as searching, parsing or decoding. Unfortunately, the size of this search space is usually huge, since the number of possible segmentations grows exponentially with the input size. Nevertheless, the number of different segments only grows quadratically so, when the generation of a segment does not depend on the generation of the other ones, it is possible to build a graph to compactly represent all possible segmentations. This information can be combined with another model which estimates which label sequences are, a priori, more likely to appear. The result of this combination allows the computation of the sequences of labels which best "explain" the observed signal. This is, roughly speaking, the approach known as "two stage". In some special cases, the models used for computing the sequence of classes, and the models required to compute the likelihood of observing a class given the segment make it possible to employ a "one pass" decoder. These search spaces can also be explored using techniques other than dynamic programming (e.g. "stack decoding"). Finally, a sole or different techniques can be applied in multiple passes: the first pass is usually done with computationally cheaper models in order to narrow the search space to be further explored by more complex and expensive models.

Most work in this field is focused on quite specific areas, so it is sometimes difficult to distinguish which parts of the proposed solutions are task dependent and which ones are more general. What seemed, at a first glance, different problems may sometimes be solved with similar techniques. Indeed, the similarity observed by [Manguel 1998] between the loud reading and speech[4] can be translated to the pattern recognition domain where both speech recognition and cursive handwriting recognition are often tackled with the same techniques based on Hidden Markov Models (HMMs). These models have become so successful that most of the work in that field seems Goldberg variations on the same HMM theme. HMMs suppose that observed sequences are described in terms of another hidden sequence or underlying units. The idea of spoken words composed of a sequence of phone

4 Indeed, even when doing silent reading, a phonological representation of read words is made and the corresponding neural areas (related to speech) are activated which can be explained by a recycling in the use of some areas [Gibrat and Dehaene 2012].

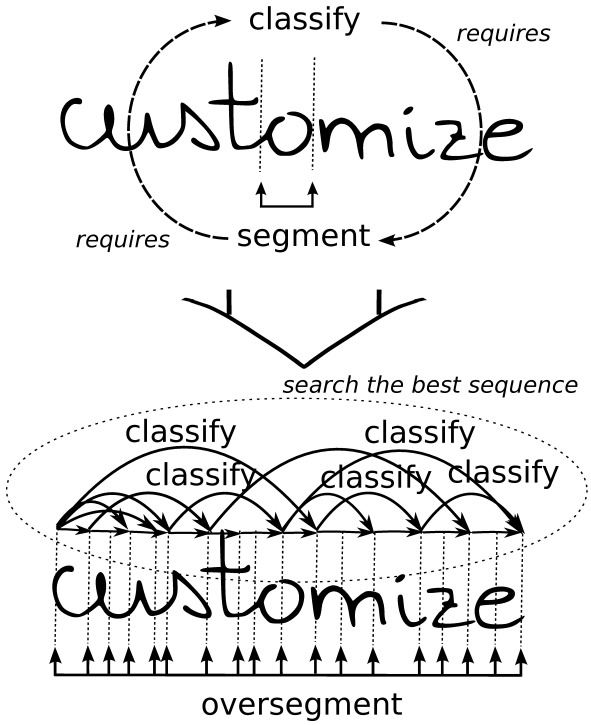


Figure 3: A possible way to break Sayre's paradox circularity by means of over-segmentation. As a result of over-segmentation, lots of possibly wrong segments have to be classified. This information, relating the likelihood of image fragments constituting letters, is combined with information about the a priori probability of letter sequences in order to obtain a huge search space. Finally, the best hypotheses are chosen from this search space.

segments is graphically referred to as beads-on-a-string paradigm [Ostendorf 1999]. Obviously, these models are crude approximations of reality. We do not speak by short bursts of sound from a set of independently distributed probability function nor by concatenating uncorrelated phone segments without any kind of co-articulation. Likewise, the same observation can be done relating handwritten recognition where it is neither possible to consider a cursive handwritten line as the concatenation of image segments associated to individual graphemes. The image of a handwritten line may contain, in the same column, several strokes corresponding to different graphemes. Moreover, a sole stroke may correspond to several letters with no clear separation between them. This mismatch between models and reality can be partially reduced by using context dependent units and by adding context into the input features.[5] Sometimes, simpler and more inaccurate models may perform better at recognition, when trained with

5 There may exist a correlation between nearby input features in HMMs. What HMMs assume is that they are independent *given the hidden states*. Otherwise stated, each observed frame comes from an independent outcome or thrown of a probability density function. See Chapter 7 for more details and, in particular, for the remark made by [Bridle 2004] which consists in viewing HMMs with dynamic features as an *unnormalized generative model*.

a limited amount of data, than more accurate models which require much more data. HMMs are called "segmentation free" because they are generative models where the contribution of every possible segmentation is taken into account. Nevertheless, decoders usually follow the Viterbi approximation which ignores all but the most likely segmentation. Thus, no explicit segmentation is required to decode and an "internal segmentation" (to distinguish from explicit or "external segmentation" used by the approach known as "segmentation-and-recognition") is obtained a posteriori as a byproduct of recognition.

main aim

The main aim of this thesis is related to the study, classification and modeling of problems which can be represented in a one-dimensional way and which require a joint segmentation and classification task to be solved. More specifically, we are interested in problems which can be modeled by sequences or signals which can be explained in terms of another underlying sequence made of units taken from a finite repertoire (for instance, phonemes in speech) so that a sequential correspondence between the original signal and the underlying sequence of units is preserved (i.e. there is no reordering). This kind of problems is also known as "sequence labeling" in some contexts, but the precise meaning of this expression is not clear,[6] for this reason we prefer the longer term "joint segmentation and classification of sequences".

over-exploited niche

Our subject of study is, no doubt, a problem studied for decades by a wide and heterogeneous community. Not surprisingly, the law of diminishing returns [7] also seems to apply here and more and more sophisticated and resource intensive methods are required to barely improve the results. This can indicate that the field is exhausted or that the field is mature; so, a priori, it does not seem a good idea to look in this so over-exploited niche for new contributions. To put things even worse, we will focus our work in the area of decoding: It has been demonstrated that optimal decoding is a NP-hard problem [Casacuberta and de la Higuera 1999],[8] whereas the technology that predominates in most practical applications, HMMs and Viterbi decoding, is linear with the length of the processed signal,[9] and this seems to be a lower bound. Thus, a naive thought could be

> *Why get worried about decoding? After all, progress explained by Moore's law will improve our current systems even with no real innovation. Implementing a new decoder requires a big effort that will, in the best case, give us today a decoding system similar to a current system tomorrow. Are we really pretending to do it better than all this clever people who tried that before? Are not we going through a minefield of "wheels to be reinvented"?*

6 For instance, wikipedia entry `http://en.wikipedia.org/wiki/Sequence_labeling` define sequence labeling as the *"assignment of a categorical label to each member of a sequence of observed values"* assuming a 1-to-1 correspondence, whereas, for instance, [Graves 2008] uses this expression to refer to *"all tasks where sequences of data are transcribed with sequences of labels"*. A more detailed classification is provided in Chapter 3.

7 Wikipedia entry `http://en.wikipedia.org/wiki/Diminishing_returns` define it as the *"progressive decrease in the marginal (per-unit) output of a production process as the amount of a single factor of production is increased, while holding the amounts of all other factors of production constant."*

8 I can't understand why [Lyngsø and Pedersen 2002], with similar claims, do not cite it.

9 In the common case when a one pass decoder processes a frame sequence by using regular models. Cost will be addressed in detail later in this work.

Although a lot of previous work means baselines and references to compare with, the difficulty to replicate experimental conditions from other authors is well known, since the slightest difference in these experimental conditions could cast doubt upon results. The use of standard databases may alleviate this, but when we discuss decoding performance and not only classification rates, the problem is amplified and it should be convenient to replicate algorithms we want to compare with. However, this would mean a lot of effort and would be under suspicion of "implementation bias".

Despite this not very encouraging scenario, we have chosen to see the glass half full and to believe that there is still room for improvement relating to this topic in general and to decoding in particular. It is worth trying it: new tasks appear while old ones make their own advances, some technological advances are no longer transparent from the programming point of view. Therefore, decoding has to address new demands and opportunities.

Remembering the quote[10] "dwarfs standing on the shoulders of giants" the point of this work is not at all about changing paradigms but about providing some contributions to what has already been done, hence the surname of this work "two cents on decoding". The risk of reinventing the wheel is very high but it is minimized by means of a thorough endless study of the state of the art, and the work of summarizing all that huge mess, despite not being as attractive as opening new frontiers, is quite useful as well.[11] Finally, we are quite proud of having been able to develop a cutting edge toolkit *ex nihilo* (from an empty scratch *Emacs* buffer). Let us enumerate some reasons and new opportunities which might explain or justify why new advances can be envisaged:

RELATED TO DECODING

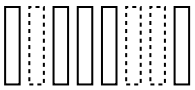
- sometimes different decoding algorithms can be studied into a new perspective which renders their similarities more evident. For instance, [Godin and Lockwood 1989] showed that One Pass [Vintsyuk 1971; Bridle *et al.* 1982] and Level Building [Myers and Rabiner 1981] algorithms are essentially identical when syntax constraints were applied [Young *et al.* 1989];
- semiring computing also provides a theoretical framework which may put different parsing and decoding algorithms in a common setting [Goodman 1999; Mohri 2002]; $\langle \mathbb{K}, \oplus, \otimes, 0, 1 \rangle$
- minimum word error rate decoding [Evermann 1999] constitutes an improvement regarding the mismatch between the assessment and decoding criteria. For instance, in a dictation system the subjective quality of the recognition depends on the number of words the user has to manually correct (which is related to the Word Error Rate (WER)) and not on the number of sentences

10 Although it can be traced to Bernard of Chartres (12th century) and many others, it is more popularly attributed to Isaac Newton.

11 "huge mess" does not pretend at all to be a derogatory appreciation. We have sometimes prioritize exposition of ideas over enhancing our own contributions in order to make easier to get a better insight on the field. Moreover, due to the large number of claims that we have found in the large vast literature which, we believe, are reinventions of the same set of wheels, we have to be very prudent relating our own claims.

in which the errors are located (measured by the Sentence Error Rate (SER)) which is the criterion commonly used to train the models (maximum a posteriori criterion);

- the number of human interactions has been proposed, in the field of interactive pattern recognition, as a measure of performance. [Oncina 2009] proposes an strategy to provide a new solution after each feedback cycle which differs from others [Toselli *et al.* 2010] and which has been generalized more recently [Alabau *et al.* 2012];
- by incorporating confidence information into the decoding process, the recognizer may be directed to the most promising paths [Abdou and Scordilis 2004];



- it is possible to take the frame-level uncertainty caused by noise into account during the decoding process. For instance, finding the "best" state sequence allowing the possibility of excluding a maximum number of observed frames which are considered as possibly corrupted [Siu and Chan 2006];
- Language models (LMs) are closely related to decoding since they contribute to define the search space. Most decoders used in this field are based on finite state models and many decoders are tuned to a particular type of language model.[12]
  Context free languages constitute a broader language family that can be taken into account when designing decoding algorithms. The use of these models is not new, but we believe some contributions are yet to be obtained and more insight can be provided relating the relationship between different models which are essentially similar but are traditionally described in very different ways. Moreover, we believe that many features such as interactive transcription, imperfect transcription or error correction can be formulated in terms of language modeling. Also, new contributions in the field of language modeling can be envisaged.

RELATED TO THE TASKS

- spoken dialog systems requires spoken language understanding, but we would like to have interactive systems able to understand at the same time the user is producing his/her utterance. A person may interrupt the speaker when all possible meanings of the sentence to be completed makes it convenient to take an action and politeness circumstances allow the interruption, so why might not a machine do the same thing? Even if we do not interrupt, people usually give "early feedback" in form of facial expressions that virtual human dialog systems should also imitate [Sagae *et al.* 2009].

  The only reasonable way to allow a dialog system to do that consists in beginning the understanding stage as the user is speaking, mixing recognition and understanding accordingly instead of following the more classical sequential or cascaded approach. The production of partial hypothesis before finishing the utterance has been addressed in decoding in many ways: reliable

12 Even for n-grams with a particular n-gram order!

identification of spoken words before the end of their acoustic realization or "early recognition" [Scharenborg *et al.* 2007], determining when a partial hypothesis has a validated prefix (partial trace-back) [Brown *et al.* 1982; Imai *et al.* 2000], to mention a few;

- it is not possible to obtain a perfect automatic transcription in most tasks. In those cases, recognition systems can be useful to reduce the human effort to correct the automatic transcriptions. This approach is known as interactive assisted transcription. But interactive tasks are very sensible to the response time in a non-linear way: when a user is working in an interactive transcription system, he/she can feel happy with a delay of 0.1 seconds, can be annoyed if this delay reaches 0.3 seconds and he/she will simply refuse to work when it exceeds 1 second. Surprisingly, some works devoted to these tasks do not report quantitative values relating the time response of their systems.

  This type of tasks is very interesting from the decoding point of view: they usually handle multi-modal and partial information (in form of user feedback), and they could benefit from confidence measures which could be graphically displayed in order to increase the visual feedback. Also, validated transcriptions can be used to improve the models by increasing the training data or by performing user adaptation. Decoders are faced to very severe response time restrictions. Ideally, the system should *anticipate* the user intentions even before he/she makes explicit feedback behaviors (for instance, by using eye-gaze tracking or by tracking the stylus approaching a given area) to perform some speculative or anticipatory decoding in order to reach a zero time response;[13]

- preprocessing is very task dependent and most preprocessing stages require taking hard decisions from soft estimations. For instance, in handwriting recognition, the binarization step can be avoided if normalized gray level (enhanced) images are used during feature extraction. But not all hard decisions can be so easily avoided: during a non-uniform slant correction stage, a slant angle must be chosen for each column. Generally, an error after a hard decision is much more difficult, if not impossible, to recover. A way to alleviate this fact may consist in using non-determinism at the preprocessing stages in order to delay these decisions to later stages where more information sources will be available. Delaying this non-determinism may affect the decoder: it can explicitly take a preprocessing parameter into account, as is done with the vocal tract normalization by the MATE decoding algorithm [Miguel *et al.* 2008]. We will try to show that directed acyclic graphs (DAGs) can be used to express this non-determinism, in a more general way, by both one pass and two stage decoders and also that feedback information is possible from the decoder back to the earlier stages;

13 Other idea not related to decoding but also useful to further improve interactive transcription systems consists in easing the readability of the scanned text. These kind of techniques on handwriting are proposed in this work.

- indexing and transcription of broadcast news, or any other data source where a partial transcription or summary can be provided, is not so uncommon. Decoders can use, in general, partial or imperfect transcriptions to guide the search [Lecouteux *et al.* 2008] although, as we have just pointed out, this information source can also be handled by means of language modeling techniques making unnecessary to adapt decoders.

RELATED TO THE TECHNOLOGY

- during many years, processors were getting faster and faster, so that the same source code led to faster decoders "for free", i.e. without any programming effort. Now, the situation can be summarized as follows:

  > *Moore's law still applies but it is no longer transparent to the programmer, who has to resort to parallelism and to special hardware features to take advantage of multi-core platforms.*

  Besides parallelism, it is mandatory to take the well known memory hierarchy into account in order to improve performance. In this way, we have developed several *cache oblivious* algorithms and we have experimented with language models stored in secondary memory. Among the technological advances that must be considered nowadays, we have to mention distributed systems and graphical processing units (GPUs);

- but we are not only interested in the most powerful machines. Embedded devices, like smartphones, play an important role in our society. Many applications related to speech or to online handwriting recognition are specially relevant in this setting. Some solutions, such as those proposed by Google on Android or by Apple on its Siri software assistant, bet on the client-server approach: speech recognition systems are placed on central servers.[14] But it is clear that a local decoder in the device would also have some advantages: no need to dispose a network connection (more robustness and resilience) as well as some chance of privacy.[15] In that case, power consumption and memory footprint impose severe restrictions;
- in the era of big data, some decoding tasks, such as video and audio indexing, comprise a very huge volume of information.[16] The energetic cost of decoding tasks is of great concern since the tiniest improvement has a great impact in absolute terms. Trying to reduce the number of watts when decoding at "a given performance level" is not at all the same aim as reducing latency. Besides, a large scale recognition system on a distributed server farm may be very different from that of a single computer. The "ideal decoder" would reduce latency, memory footprint and power consumption at the same time. But, are those objectives opposite or compatible?

14 At least at the time we are writing this work.
15 Not assured even in this case.
16 In 2010 "Every minute, 35 hours of new content were uploaded to YouTube" `http://youtube-global.blogspot.com/2010/11/great-scott-over-35-hours-of-video.html` and now this value is incredibly higher.

1.2 THE SUCCESSFUL TRILOGY

From all these reasons, we believe that decoding should be studied in context and that we should not forget at any moment the whole picture and the main aim: to improve the overall system. One of the best ways to express this can be found in [Aubert 2002]:

> *Given the many inter-dependencies existing in a large vocabulary continuous speech system, it seems that the best performing systems are almost always the results of a so-called successful trilogy, namely, that they are the careful outcome of the following contributions:*
>
> 1. *a number of powerful algorithms (not just one!);*
> 2. *a clever design and implementation cooperative with the hardware;*
> 3. *a careful tuning of all heuristics (pruning thresholds, penalties, scaling factors).*

The "*successful trilogy*" is a central theme or *leitmotiv* in this work: instead of bringing a specific contribution, we have tried to take the three points of the trilogy into account to improve results.

What has been done:

Relating formalism, we have...

- reviewed the formalization of two stage generative models (TSGMs). We have also related this formalism with other ones such as dynamic graphical models, some segment models proposed in the literature and some approaches based on recurrent neural networks to estimate frame posteriors;
- studied some algorithms required to explore the search space of TSGMs, the relationship between "one pass" and "two stage" decoders making it clear that the one pass can be applied to a restricted subset of models. In this way, we have decided to follow the trends known as "semiring computing" and "parsing as intersection" proposed by several authors in order to formalize the process of parsing (up to context free models), by means of language equations, as the computation of a value;
- we have also been interested in how decoding methods can be adapted to deal with imperfect transcription and assisted transcription tasks. To this respect, we have basically shown that some of these features can be expressed by means of language modeling techniques;
- explained how different types of over-segmentation information can be generated by preprocessing techniques (proposing a new type of over-segmentation mark) and how this information can be used by (both two stage and one pass) decoders to prune the search space. The effect of this pruning is not only aimed to reduce the running time but, surprisingly, may also improve accuracy.

Relating implementation

- provided a set of specialized decoders which can be combined in flexible ways by means of general APIs;
- we have also implemented a dataflow architecture together with a DAG serialization protocol in order to obtain several recognition systems from a set of basic dataflow "lego" pieces;
- some specialized decoders which take profit of specialized topologies to behave more friendly with the cache have been designed, proposed and implemented. Some of these decoders can entirely avoid the use of hashing techniques, they have been parallelized in order to take profit of multi-core processors and could take profit of specialized hardware architectures.

Relating specific tasks

- developed preprocessing techniques for offline handwriting recognition (HTR);
- validated them with IAM offline database by using HMM/ANN models;
- tried an external over-segmenter at word level;
- lexicon free approaches;
- right-to-left decoding;
- combination of hybrid HMMs (using ANN to compute emission probabilities) with discriminative segment (holistic) classifiers;
- bimodal recognition.

What has not been done: we have not...

- reported detailed results on ASR tasks in spite of the fact that our decoders have been used to this end. We have even implemented speech preprocessing modules based on [Nadeu *et al.* 1996] and tested with Media [Bonneau-Maynard *et al.* 2005] and with the Spanish monolingual News from WMT2011 [Callison-Burch *et al.* 2011];[17]
- studied or taken into account some entire families or decoder types, notably those based on the static network approach (with is usually based on on-the-fly composition and transducer determinization algorithms) or on stack based decoders ($A^\star$). We believe that a comparison with those models would substantially improve the significance of conclusions;
- reviewed model estimation. We have just made brief references to the techniques used in our work but we have not performed any review nor proposed novel ideas;
- implemented and validated experimentally all the proposed algorithms: Although many of them have been actually implemented and tested, lack of time has prevented us to validate, at least, the following ones:

17 These results, reported for instance in [Zamora-Martinez *et al.* 2012], are not detailed in the experimental part of this work.

  - we have proposed an approach to deal with imperfect transcription by using special LMs, but we have not validated them empirically for the moment;
  - although some bimodal recognition experiments have been performed, they have been limited to isolated word classification. Although we describe how bimodal (in general, multi-modal) continuous recognition (i.e. at sentence or paragraph levels) can be performed in two stage approaches by combining the DAGs generated in the first stage, we have not tried them experimentally;
  - we have explained how context dependent units are handled by decoders through this work, even for across word context dependent modeling, but we have only conducted experiments with context independent models, mostly because context independent HMM/ANN, when the connectionist part receives a relatively large context, may obtain very good performance measures;
  - we have described an algorithm to construct DAGs from another DAGs. It can be used to obtain a DAG of words from a DAG of phones or graphemes, but it has not been fully implemented and tried on HTR or ASR tasks;
  - we are interested in the use of context free LMs in the form of recurrent transition networks (RTNs). We have devoted a large part of the first, more theoretically focused, part of this work, but no experimental results validate this extension, mostly due to the lack of the proper tasks and models. This is one of our priorities in the future work.
- we have not completely profited from one of the main advantages of the two stage approach, namely, its capability to know in advance the beginning and the ending points of the segment whose likelihood is to be estimated. HMM/ANN have been used in most tasks and, although the use of discriminative segment models has been used in the form of some holistic ANN approaches to HTR, the use of this type of segment model in combination with generative approaches (as described in Section 7.7) has been developed but more experiments would be required to take a greater profit of this proposal.

... and what should be (have) done (possibly by others):

don't bite off more than you can chew

Note that we cannot be too much ambitious: The combinatorial explosion of tasks, preprocessing and feature extraction techniques, selection of elementary units, of techniques for modeling those units, the influence of context, the possibility to limit this context within the lexical items where they appear or also across them, not to mention that the presence of a lexicon can sometimes be questioned, the huge type of LMs and methods to estimate them, the items that can be obtained at the output of the system (e.g. just the best sequence or the best ones? a confidence measure? is it possible to reject?),... this list is *so huge* that it has not been completely explored by the research community even after decades of work.

A way to deal with that would be to emulate Ramon Llull's circle[18] in order to *systematically enumerate* all those possibilities, establishing a hierarchy and a relationship among models and combinations. This effort should be accompanied by a literature revision in order to check what has already been done, to normalize the inconsistent and sometimes contradictory nomenclature and to trace these results in a map as in some previous review works [Aubert 2002] but at a considerably larger scale. Besides drawing a map and tagging some "points" with drawing pins (recognition systems, publications,...), some criteria should be provided or even mined to determine which ones are more suitable for a given task under some particular conditions. We will go into detail about these ideas in the following section.

1.3 SOME REMARKS ON THE APPROACH TO DISCOVER NOVELTIES

Previous remarks about the combinatorial explosion of techniques makes us to think not only about how to better disentangle what are better innovations, but also about the necessity to have better ways to manage the constantly increasing number of published papers at our disposal. This section contains some remarks about the approach we have tried to tackle in this area of research, where so much has been already done in decoding, in order to heuristically detect where novelties can be obtained as well as to express our humble opinion on the interest of organizing the overwhelming number of results described so far in the literature. The purpose is to justify a part of the work which has been done anyway, as part of the development of this work, even if it were not reported here.

Given the maturity of the field, we are convinced that a necessary step, which can be seen as a desirable product of the research work, is to structure the knowledge by organizing a joint space or "map" of tasks, problem types, models and solving techniques as well as enriching it with advice on which techniques are more suitable for different combinations of features and constraints.

levels of innovation quality

From this perspective, it is possible to classify the contributions in different levels of quality,[19] from higher to lower, as follows: In the highest level of innovation we would place contributions that modify the shape of this map instead of just pointing new drawing pins into it. This modification can be performed in different ways:

- by expanding the map. For instance, by discovering or inventing new models, algorithms, representations or preprocessing techniques;
- by folding the space demonstrating that different models, tasks, etc. are essentially the same;

18 Which inspired the *Ars Combinatoria* by Leibnitz. A similar idea is the Musikalisches Würfelspiel system.

19 This classification of "innovation levels" may be just a subjective opinion for this engineering field which, in any case, cannot be necessarily extrapolated or applied to other areas.

- by changing the shape making it more structured. For example, by converting simple enumerations of techniques into more understandable combinations of features, demonstrating that some configurations are just a combination of previous ones.

The following level in our particular classification of novelties from higher to lower quality is the exploration of this joint space, what we have metaphorically described as putting a drawing pin. For instance, when one article claim to be "the first one trying a particular combination of features", as was the use of HMMs in handwriting after showing its usefulness in ASR. The result of this exploration, the equivalent of putting drawing pins into the map, should help to better determine which combination of models and problem solving techniques are best suited for some particular tasks and problem types and, hopefully, to extrapolate these results to configurations never tried before to obtain a more informative way to further explore this search space. Note also that this is not the aim of most researchers which just try to solve a particular task.

reproducibility of experiments

The complete specification of an experiment, which should give rise to its reproducibility by others, includes taking also into account what researchers would qualify as mere "tricks", a know-how which is not often reported in the literature into too much detail for several reasons:

- it is widely accepted that these more engineering aspects (implementation details) are much less interesting compared with the scientific or theoretic counterparts;
- we risk to seem dummy, maybe these things are not explained because everybody should already know them;
- the space in scientific papers is an scarce resource: the number of pages being limited in most conferences, we have to prioritize and reserve it for more important things, specially because details are much more numerous. Besides, trying to detail too much may overwhelm the readers and is opposed to the principle of highlighting the essential. Other times they are ignored just because it is too boring or are just taken for granted and, hence, ignored;
- for a completely different reason, keeping these details is like preserving a precious know-how, some kind of industrial secret.

novel vs established techniques

Relating the empirical validation of some techniques for some particular problems and tasks, a peculiarity of this field is the fact that, quite often, a wrong model (e.g. a probabilistic model incorrectly derived) or a bug (e.g. in a parametrization) do not necessarily imply a bad result, it can even perform reasonably well! Indeed, in many cases the pragmatics come before the theoretical explanations. On the other side, it is reasonable that novel techniques obtain regular results since they have not been tuned as much as well established techniques [Bourlard *et al.* 1996]. This phenomenon is in conflict with the observation that the law of diminishing returns makes more and more relevant minor improvements obtained on the more elaborated approaches. It is not easy, in this scenario, to fairly evaluate and compare paradigms outside the mainstream.

Unfortunately for the human ego, this exploration is often related to a systematically exploration and to routine work, as graphically illustrated with the Lullian Circle technique.[20] Nevertheless, the number of combinations being so huge, the exploration is not blindly performed in a systematic way or even by means of techniques based on the design of experiments but, rather, it is more guided by heuristics, intuitions or explained by socioeconomic factors such as the inertia on the research interests and the background of people working in specific research group or enterprise, the presence of datasets for some tasks, as well as other types of social factors including how the work of researchers is measured in terms of a weighted number of published papers, the presence of trending topics in the entire area, economical issues, competitiveness and a large etc.

The lowest level on novelty would be to organize the already existing knowledge, in such a way that it will hopefully provide some new insights. In this way, it is possible that most readers will not understand the interest in the first part of this writing, where probably more effort than the strictly necessary has been devoted to propose a particular view of the state of the art, which, in addition to bore some readers (in some cases), could be replaced by a properly selected set of bibliographic references to reviews.[21] After all, it is not worth making the effort when this activity is considered as having a lower level of innovation and an lower expected return of results w.r.t. the invested effort. For reasons of opportunity cost, it would be a better idea to use this time to rush ahead rather than to review back.

Relating the question of why not to simply replace the first part of this work with a selected number of well chosen bibliographic references, one of the reasons was that, in many cases, we have not found in previous literature things organized in the same way as we saw them. For instance, most reviews and even introductory books seem an agglomeration of topics where each one has been written independently and where focus is on the most popular techniques ignoring the more marginal ones.[22] Another reason is that our so very limited attempt to make our personal version of this "map" does not correspond to any pre-established plan (since this would be unfeasible) but it is rather the byproduct of organizing personal notes while reading papers during the realization of this work[23] and, maybe, a personal deformation caused by years of teaching.

Some readers may find some critics or remarks to some prior art in this part of the writing a little *unfair* because we have analyzed some works specially devoted to specific tasks (e.g. many works dealing specifically with ASR) from a perspective different from the scope of

20 Some patents and some papers describing just these automatic/mechanical/formal combination of previous inventions and ideas would be just prior art from this point of view. Obviously, this depends on how easily anyone faced with the same requirements as the authors of these works would point to the same conclusions. Our claim is that organizing or structuring the knowledge instead of just considering it as a set or log of publications should raise a lot the leveling rod dividing what is to be considered a real innovation.

21 Indeed, maybe the excessive number of bibliographic references is another plausible critique to this writing.

22 Is like finding a phylogenetic taxonomy in XXI century talking about bacteria and eukarya but ignoring archaeas as a main domain only because they seem marginal.

23 The personal notes largely surpass the size of this writing.

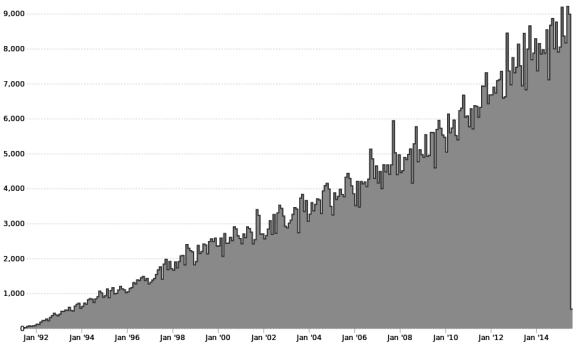


Figure 4: Monthly submission rate to `arxiv.org` over around 24 years. The total number of articles available is 1,061,328. (Source: `http://arxiv.org/show_monthly_submissions` consulted on August 2015).

original authors and intended context in order to deal with a generalization which takes more tasks into account. The same remark could be done, in the future, to our work by someone else interested in a broader scope than ours, as is the case of statistical pattern recognition on structured data not limited to sequences.[24] For instance, we have studied the recognition of handwritten lines and it is widely known that handwriting includes layout analysis and much more stages than those investigated here.

The idea of organizing a joint map might contrast with the current knowledge manipulation system based on feeding an increasing number of conference papers and journals (exemplified in the illustration of Figure 4). This publishing system has been often analyzed and even criticized for several reasons, including the discussion of effects such as the publication bias[25] or how to properly evaluate the novelty of contributions, etc. Our aim and role is not to criticize it or to question its purpose to transmit "packets of innovation" where new ideas are proposed and new results are reported.[26] The problem is the use of this "log of transmitted packets" as the main resource for learning what has been done so far, since this produces a non-linear increase of effort over time when new researchers are introduced into the field. This situation remembers us an example proposed to students of algorithms to explain quadratic costs:

> *A painter is marking road lines with a brush without moving the paint can which remains at the beginning. The more she/he progress, the more effort she/he has to invest to obtain more paint.*

Researchers trying to explore the increasing number of prior art have to select a limited subset of publications despite the existence of topics to classify papers, keyword lists and the use of search engines,[27] which

24 More particularly, to sequences explained by an underlying hidden label sequence in a monotonic way.

25 See, for instance, `http://en.wikipedia.org/wiki/Publication_bias`.

26 There exists continuous debates in the scientific communities about the publication models, e.g. `http://nips.cc/Conferences/2009/PublicationModelsDebate`.

27 I have discovered many relevant contributions which were not related with the title of the paper or with the main claims, which makes me wonder how someone who was

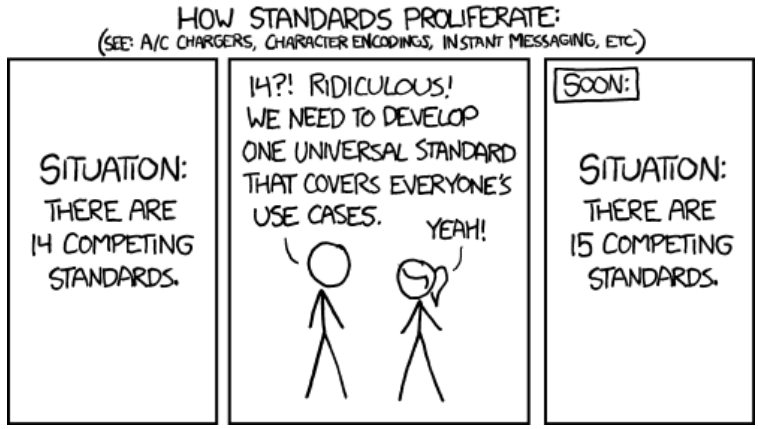


Figure 5: Proliferation of standards (Source: `http://xkcd.com/927/`).

information overload

may explain why things are sometimes forgotten to be reinvented later. Despite its low level of innovation, structuring existing knowledge is also useful at least to avoid duplication of efforts, to help to disentangle what are real innovations and, why not, to provide better learning material for new students and researchers. In this way, survey papers such as [Deng and Li 2013] constitute an invaluable guide. Note also that it would be interesting a radical idea that will probably never be followed: the use of ideas similar to distributed *revision control systems*[28] to convert current scientific contributions (papers seen as packets of novelty) into manipulations of this joint space more structured than the current set of competing journals organized into broad topics, i.e. the use of (using the nomenclature of distributed systems such as *git*[29]): issues, branches, merges and push requests.

nomenclature overload

We have also detected too much overload of nomenclature as if assigning a new name to a particular combination of features would make it seem deeper.[30] Contrarily, generic terms may sometimes refer to specific cases: in most works the term "HMM" implicitly means HMM/GMM making it difficult to use an unambiguous term to talk about general HMMs no matter the emission type. Similarly, the expression "segment model" usually denotes something different from HMMs which, in our opinion, are a particular case and not the opposite of segment model. Nomenclature problems seem an unavoidable consequence of a working community. That is why we have not proposed any unified nomenclature and have avoided too much definitions in this work: it is not our purpose, we lack authority and this probably would make the situation even worse, as depicted in Figure 5.

We have tried to generalize some expression from task specific concepts to the general model, although it is quite difficult to avoid coined terms such as "acoustic model" which has different equivalents in

looking for this particular feature could ever have find with current searching engines (I cannot foresee what will happen with those yet to come).

28 See `http://en.wikipedia.org/wiki/Revision_control`. A similar idea for the generalization of the hierarchy of problem types is proposed in Section 3.3.

29 See `http://git-scm.com`.

30 Some particular cases of HMM have more surnames than most royal families!

HTR.[31] Note that there is an additional risk when trying to generalize some coined concepts, as illustrated from this quote:[32]

generalizing coined terms

> *don't be surprised when people find your results counter-intuitive, because you are using a term that they already understand in a way that doesn't conform to their understanding.*

To conclude this section, let us remark that the very idea of a joint map with drawing pins is a particular point of view which may provide some help but would also constitute a limitation and other different approaches are required. For example, the way this map is constructed constitute an important bias, related to the *linguistic determinism*.[33] Although it can be considered as a goal or as the final product describing the result of research (something to be obtained a posteriori), we believe that this may be *also* considered as an interesting way to detect novelties easier in this exploited niche and we have profited of this in some cases through this work. Even if trying to follow this approach has not the tiniest chance to achieve any accomplishment, we believe this constitute a nice *heuristic* to easily *detect* were to look for novelties and to easily filter the more straightforward ones, since anyone who is friend with his/her brain and who knows that 2+2 equals 4 would probably reinvent something when faced to the same problems under the same conditions. We have tried to list what we think are new situations and conditions, it now only remains to draw the map, put drawing pins into it in order to look for empty areas in the hope of being able to find something beyond the mere mechanical combination of previous ideas. We will try to show that some parts of this work correspond to novel techniques.

1.4 ROAD MAP

The remainder of this writing is mainly organized into three parts (which roughly correspond to the three points of the successful trilogy) divided into chapters as follows:

1.4.1 Problem statement and formalization

CHAPTER 2. SOME APPLICATION DOMAINS are very *briefly* described in order to give an informal idea of which kind of problems could benefit from the techniques and models described in this work. The motivation is twofold: using several examples makes more difficult to confuse ad hoc properties with general ones. On the other side, some examples show how preprocessing ambiguities may appear, for some problems, in the very first preprocessing stages. The use of explicit representation of ambiguities alleviates what is known as the "pipeline problem": the

31 Some authors propose the term "optical model" and others "morphological models" to describe the handwritten recognition counterpart of "acoustic" model.

32 Extracted from a discussion about the meaning of parsing from `http://compgroups.net/comp.theory/mark-w-hopkins-theory-perspective-on-parser-engin/1131099`.

33 Linguistic determinism is the idea that language and its structures limit and determine human knowledge or thought.

requirement of taking "hard decisions" between some stages of a pattern recognition system. Also, some ideas related to preprocessing are proposed which will inspire the techniques described in Chapter 14 as well as the different underlying representations (frame, segment and landmark based) which will be convenient when discussing about inputs and output types in the following chapter or when discussing about segment models in Chapter 4.

CHAPTER 3. PROBLEM FRAMEWORK briefly reviews some types of supervised pattern recognition problems on sequences without considering, for the moment, which recognition techniques are best suited for the described problems. They are, rather, characterized in terms of the different types of inputs (including external segmentation information) and outputs (including an extensive classification of output types). We have tried to deal with some extensions to these problems such as the presence of several input sequences or the case of interactive transcription. We also discuss the limitations of this classification in terms of inputs and outputs and envisage the possibility of representing these type of problems (including interactive assisted transition) in an homogeneous way using ideas inspired by the "dynamic annotation framework" which would explicitly include (a reifying of) the processes performed by different types of actors (automatic systems, human transcribers, and so on). The following section is devoted to analyze the features and properties which may relate problems different on surface, trying to forecast which questions could be posed, when faced with a new problem. A final section addresses the issue of how to analyze the behavior of recognition systems and how their performance can be measured.

CHAPTER 4. TWO STAGE GENERATIVE MODEL (TSGM) is a doubly stochastic generative process where the observed signal is generated in two stages: first, a sequence of units is obtained. Secondly, every unit from this sequence is used to generate/explain a segment of the observed output. This model includes not only the well known HMMs, but also *some* types of segment models (SMs). A hierarchy is described with regard to the independence assumptions associated to the second stage. A review of graphical models (GMs), including dynamic GMs and their relationship with TSGMs is also provided.

The three "classical problems" (namely: probability of observation, estimation and decoding) are briefly addressed paying attention to decoding: a search space can be obtained by using the generative model in a *reversed way*, using non-determinism when required. This leads to the "two stage" algorithm where a DAG is obtained from the observed signal to be later combined with a language model (LM). The relationship with the "one-pass" algorithm and with over-segmentation are also discussed.

For the sake of completeness, some alternatives to TSGMs are reviewed, including models based on segment-based observations, others based on frame posteriors and on the use of Connectionist Temporal Classification (CTC) decoding, etc.

CHAPTER 5. RECOGNITION, DECODING, PARSING, TRANSLATION is aimed to formalize and to describe the process of parsing and decoding required to combine the DAG described in the previous chapter with a language model (up to context free models). We have followed the approaches known as "semiring parsing" and "parsing as intersection" in order to describe this process as the computation of a value. Although these models are usually formalized in terms of rewriting systems, we have preferred to use systems of equations. A novel transducer composition algorithm that can deal with null transitions without resorting to filter composition techniques is proposed. It is extended from regular to context free models. A special attention is paid to recurrent transition networks (RTN) for representing both the context free models (RTNs are presented a *normal form* of CFGs) and the parsing result (acyclic RTNs without recursion cycles are essentially equivalent to *packed shared forests*). The composition algorithms have been extended to RTNs and a transformation of the models into a "homogeneous epsilon" form is proposed to this end. Although the use of hypergraphs is often used in this setting, they seemed overkill and have not been required. This chapter also tries to clarify the relationship with other widely known parsing algorithms such as CYK or Earley.

CHAPTER 6. SEQUENCE/LANGUAGE MODELS is focused on the models associated to the first stage of TSGMs. This chapter is structured as follows: A classification of LMs is done depending on how the probability of generating a sequence is decomposed. The most common intrinsic figures of merit are also described. Next, some issues related to the lexical items are analyzed since they are quite orthogonal to other aspects: the most common lexical item types, the problem with open/close lexicons, their representation (e.g. factored and continuous). Another section deals with features such as combination of LMs or how to deal with changes of the external context (e.g. cache based LMs) in order to use them when reviewing the most widely known LM types. n-grams are a particular case of the "clustering histories approach" under the Markovian assumption. They are divided into count-based and feature-based. A final section is devoted to discuss how some features can be described in terms of LMs, related to interactive transcriptions systems, imperfect transcriptions, etc.

CHAPTER 7. SEGMENT MODELS addresses models aimed to estimate the probability of generating a frame sequence given a hidden/latent label starting by a description of the main limitations of HMMs and how SMs other than HMM have tried to overcome them. This chapter does not pretend to be a complete review of SMs appeared in the literature. Instead of a never-ending exhaustive list we prefer to provide elements allowing their classification, including a general scheme of frame emission. Finally, although discriminative segment classifiers cannot be used directly in a generative framework, we explain how they can be used by combining them with generative SMs.

1.4.2 Proposed algorithmic solution

The second part to this writing complements the first one by providing some algorithmic solutions. The distribution of this part into chapters is as follows:

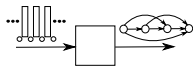
CHAPTER 8. DATAFLOW ARCHITECTURE explains how to achieve flexibility by assembling processes (kind of "lego" pieces) which are executed concurrently and which can communicate with others by means of messages sent and received through channels. A DAG serialization protocol allows the transmission of DAGs between components in such a way that they can be emitted while they are being produced and, reciprocally, consumed while they are being received (the entire DAG is not necessarily represented in memory at any moment, and most of the proposed algorithms can be described, in a reactive way). A modest catalog of dataflow components suffices to illustrate a large repertoire of recognition systems described as dataflow networks. These examples show the interest on this architecture and help us to better understand the behavior of the dataflow components to be detailed in following chapters.

CHAPTER 9. LM REPRESENTATION is a counterpart of Chapter 6 related to implementation details. An interface between LMs and decoders has been designed and a representation of count-based n-grams based on sorting states by their fan-out has been implemented. This representation can be used from secondary memory means of memory mapping.

Although NN LMs are not the main focus of this work, some improvements are proposed as the use of skip NN LMs, the estimation of normalization constants by means of an auxiliary LM and a novel type of NN LM specially designed to perform LM look-ahead that requires a dedicated decoder.

CHAPTER 10. SEGMENT ESTIMATORS addresses the design and implementation of decoding algorithms specialized in computing the likelihood of emitting a frame sequence given an elementary unit or a word from a lexicon. This chapter is the counterpart of Chapter 7, although this one is biased towards lexicon decoders. A general interface makes it easier the use of these models by DAG generators and by one pass decoders presented, respectively, in Chapters 11 and 12. This chapter also discusses the use of pruning and language model look-ahead. The interest in specialized decoders becomes clear since they take profit of some hardware features and model topologies to improve performance: for example, cache-oblivious algorithms specialized in tree lexicons. We also show how these decoders can manage hypotheses from different contexts. The combination of these ideas, leads to one of the fastest decoders proposed in this work (decoders which *do not require the hashing technique at all*, presented in Chapter 12). An extension to general lexicon networks is presented. Other specialized decoders proposed in this chapter can be used to obtain a DAG of words from a DAG of sub-word units taking into account edition operations.

CHAPTER 11. DAG GENERATION is focused on DAGs whose edges are labeled with segment generative likelihoods and how these DAGs can be obtained from a frame sequence or from other DAGs (from several ones in the case of multi-modal systems). These DAGs, used by two stage decoders, do not have to be confused with other word graphs used to compactly represent the best outputs of recognition systems. The proposed DAG generators are basically special cases of well known weighted finite state transducer composition algorithms adapted to the dataflow approach (implemented in a reactive way and based on the proposed DAG serialization protocol). They make use of segment estimators of Chapter 10. It is also shown how pruning and how the information provided by an over-segmenter can be taken into account. A *heuristic* method is also proposed to construct a DAG where vertices are clustered to reduce the size of the DAG. Finally, a DAG generator to obtain a word DAG from a sub-word based one (e.g. one labeled with phonemes or graphemes) is briefly described.



CHAPTER 12. DECODING describes both one-pass and two-stage decoders. A first section is devoted to the two-stage decoder, but we are only interested now in the second stage, since the first one was already addressed in Chapter 11.

The next section is focused on the one-pass decoder. A "dynamic" (as opposed to static composition of WFSTs) decoder is proposed. It takes advantage of the specialized lexicon decoders of Chapter 10 and maintains a representation of active states in a cache friendly way that has the particularity of never requiring explicit searches (e.g. hashing) of active states.

We have also been interested in the parallelization possibilities of decoders. The proposal for the one-pass decoder takes profit of memory shared thread-based architectures to distribute the load by using some particularities of lexicon decoders and LM estimators of previous chapters.

1.4.3 Experimentation and results

Following the rough analogy between the points of the successful trilogy and the parts of this writing, we are entering into the "pragmatic and task dependent considerations" part, where we have been mainly interested in HTR:

CHAPTER 13. LM EXPERIMENTATION The main aim of this chapter is to show some results related to language modeling. It is a complement of Chapters 6 and 9 where the ideas were presented. This chapter is mainly divided into the following parts: 1) a short description of empirical data validating our automata representation based on sorting states by their fan-out; 2) results on skipping-NN LMs in order to improve the technique of precomputing and storing softmax normalization constants; and 3) some preliminary results of the technique proposed to estimate the softmax normalization constants of NN LMs by using an auxiliary LM.

CHAPTER 14. OFFLINE HTR PREPROCESSING describes novel preprocessing techniques for offline HTR which replace classical geometrical heuristics by classifiers based on automatic learning techniques. The idea of using automatic learning from the early preprocessing stages was already discussed in Chapter 2 when explaining the pipeline problem. This approach has been followed now for HTR and it comes at some cost since the use of supervised learning techniques naturally implies some human effort to prepare the supervised data. We explain how to reduce it by means of bootstrapping, concluding that this effort seems worth doing given the encouraging results.

CHAPTER 15 HTR EXPERIMENTATION summarizes some experimental work conducted on HTR. In order to validate the preprocessing techniques from previous chapter, we have compared our preprocessing with a baseline preprocessing using the IAM offline database. Grapheme units have been modeled with HMMs where emissions are estimated by Gaussian mixtures, in one case, and with Multilayer Perceptrons (in this case, HMM are usually known as hybrid HMM/ANN), on the other side. We have also analyzed the influence of over-segmentation, at the word level, on WER and speed, we have also studied the use of lexicon free approaches to deal with out of vocabulary (OOV) words, the combination of HMM with HMM/MLP at the word level and the combination of HMM/MLP with holistic classifiers. Finally, some experiments using a bimodal corpus are also reported.

1.4.4 Conclusions and future work

CHAPTER 16. CONCLUSIONS summarizes the main contributions, establishes the conclusions and sketches the future work. During the writing of this work, we have mixed state of the art with contributions in order to favor the exposition of ideas. Now, it is necessary to make clear which things are contributions (our two cents) and to what extent they are.

Since the subject of this work is not as narrow and focused as a laser beam, the combinatorial explosion of algorithms, systems and tasks has to face with our human limitations (e.g. in time and resources). It is simply impossible to present a closed work. To this respect, works reflect not only the logical structure the authors have planned beforehand but also the chronological events and the order in which ideas have come. Plans evolve during their development and we cannot always foresee all problems and issues in advance. Some of the proposed implementations and experimentation reported here simply exist because better ones came later, so the first ones remain as "frozen accidents". We have also listed some expected critiques we are obviously aware of, but we are sure we have not been able to foresee the most interesting ones. Conclusions finish with some ideas about the future work and with a short summary of the publications derived from this PhD.

Part I

Problem Statement and Formalization

2 SOME APPLICATION DOMAINS

Puse la bombilla debajo con sumo cuidado.
Puse la bombilla de bajo consumo, ¡cuidado![1]

CONTENTS

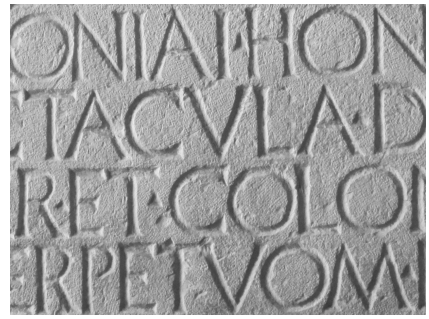
There is a risk when we center our attention on a sole problem, since we can take, by mistake, some problem specific singular features as the general case. That is the main reason why we devote a first chapter to provide several case studies which, albeit being different in surface, can be formalized in a similar way as a problem on sequences involving classification and segmentation of segments which can be modeled by statistical methods estimated from data. Hence the concepts of *front end*, which is very task dependent, and *back end* which can be roughly the same for different problems as far as they correspond to the same type. We hope that these examples will not only whet the readers' curiosity, but will also avoid us to describe later the problems types in a too abstract way (without motivating examples). The description of some problem types and some models will be addressed, respectively, in Chapters 3 and 4.

Descriptions are brief in order not to lose the focus. The experimentation part of this work will only deal with Automatic Speech Recognition (ASR) and Handwriting Text Recognition (HTR).

We hope that some examples will show how the capability to represent ambiguity during preprocessing could be useful to avoid taking hard decisions too early. This justifies the interest on using directed acyclic graphs (DAGs) or non-recursive context free models (packed shared forests [Billot and Lang 1989] or acyclic RTNs (Section 5.4.3)) as useful generalizations of sequences.

1 *"I put the lamp underneath carefully."* and *"Caution! I put the low consumption lamp."*.

2.1 TEXT SEGMENTATION



Let us begin with the same example as in the introduction: word division ambiguity when word dividers are missing. A word divider in a writing system is a glyph that separates written words. In languages using Latin, Cyrillic and Arabic alphabets the word divider is usually a blank space, but other word division glyphs exists: for instance, Ancient Latin scripts used a small dot known as interpunct, and other scripts used also a single, double or triple interpuncts, and even vertical lines to separate words. However, there are also several written systems without word dividers. The scriptio continua style of writing[2] was used in Greek texts and was also adopted for Latin after the use of interpunct died out [Saenger 2000]. Chinese, Japanese and Thai are among the main scripts consistently written without word separators nowadays. In some of these languages there is a strong relationship between characters and words, so the need for word separations may be less important than in other writing systems, which may explain why they have not adopted the use of word divisions.

Automatic word segmentation consists on retrieving the boundaries between words or, more precisely, between *lexical items* which are the basic elements of a language's lexicon. The term "token" is also used in some contexts, so this process is also known as tokenization. It is a convenient (and sometimes necessary) step in many natural language processing (NLP) tasks (i.e. machine translation, part of speech tagging) and it is not only restricted to word division: Morphological analysis involves dividing words into morphemes. Also, sentence breaking or paragraph segmentation [Genzel 2005; Sporleder and Lapata 2006; Shi *et al.* 2007] are also useful in some contexts.

When word segmentation is part of another problem, it is possible to generate several segmentation alternatives in order to delay these ambiguities to later stages. In this case, the process is known as *over-segmentation*. This usually involves to take local decisions over candidate word boundaries, so that the segments are implicitly defined by them leading naturally to DAG representations. For instance, detecting sentence boundaries may require to determine whether a period appearing in the text is used as a punctuation mark or is part of an abbreviation.

Segmentation can also be obtained as a byproduct of more complex problems: segments are not only delimited but also labeled. Indeed, some segmentation algorithms may label (e.g. using a lexicon) even if it is not required, as an intermediate stage. Other segmentation approaches rely on properties of allowed segment boundaries, measures of coherence of the segment content, and so on.

2 http://en.wikipedia.org/wiki/Scriptio_continua

2.2 PART OF SPEECH TAGGING

Many NLP tasks require to classify the words in a text as corresponding to a particular lexical category or, more generally, to assign a "Part Of Speech" (POS) tag. Part of speech is defined in a grammar as a linguistic category of lexical items which usually include (in English): noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection. Different occurrences of the same word may correspond to different POS tags, as shown in this example taken from [Weischedel *et al.* 1993]:

- a *round* table: adjective;
- a *round* of cheese: noun;
- to *round* out your interests: verb;
- to work the year *round*: adverb.

disambiguation

making this task another case of disambiguation. There are several methods to automatically classify POS tags and, although some of them are knowledge-based, the use of probabilistic models is the most successful choice nowadays. It is worth mentioning that we can reach an accuracy of 90 percent by merely assigning the most common tag to each known lexical item, and the most common tag to unknown items [Charniak 1993]. This is explained by the fact that many lexical items mostly represent their most common POS tag.

This task basically consists in labeling a sequence of items, so it does not involve a segmentation problem but only a "joint classification". The probabilistic approach consists in estimating the probability $P(T|W)$ of observing a tag sequence T, given the observed word sequence W, in order to take the most probable tag sequence, as explained into more detail in the next chapter. Since this labeling task is performed at the lexical item level, a segmentation is required in some cases. For instance, Semitic languages like Hebrew and Arabic allow to assign POS tags to morphemes which are concatenated to form words, making necessary to segment words into morphemes (morphological analysis) and to label them afterward [Bar-Haim *et al.* 2005].

2.3 TEXT CHUNKING

superficial analysis

Text chunking [Abney 1991] consists in splitting the input sentences into non-overlapping, non-recursive (or flat) "chunks" (phrases, group of words syntactically or semantically related) on the basis of fairly superficial analysis. The existence of chunks is motivated by psychological studies which relate them with prosodic phrase breaks: we tend to read texts "one chunk at a time". They usually comprise a head word and some modifiers. For instance, a noun phrase (NP), also known as nominal phrase, is a type of chunk composed by a head nominal word (noun, pronoun) optionally modified by adjectives or other quantifiers. Since there are several types of chunks, the process performed by a "chunker" is a joint segmentation and classification task. The process of chunking is also known as "shallow parsing" where the terms "shallow", "partial" or "superficial" are opposed to "deep" or "full" parsing.

Although chunks are non-recursive structures, they are, nevertheless, related to the syntactic parse tree of a full syntax analysis. That is why shallow parsing can serve as a useful preprocessing step for full parsing, since the former is usually faster, more reliable and easier to implement. Also, the information provided by chunking usually suffices for many practical NLP tasks such as information retrieval where chunking can be used to restrict indexing to base NPs.

Many chunkers reduce the joint segmentation-classification problem to an apparently simpler problem of labeling or joint classification. To this end, a labeling *scheme*, which simultaneously encodes segmentation and the type of chunk of the segment, is used [Ramshaw and Marcus 1995; Section 4]. There exist several ways to perform this encoding, the most popular is known as BIO or IOB which stands for 3 types of tags: 'B' is used to mark the first word of the chunk, words inside chunk are marked 'I' and, finally, words outside[3] are marked as 'O'. There is other variant called BILOU which adds two tags to label the last tokens of multi-word chunks (with 'L' tag) as well as Unit-length chunks (with 'U' tag). Surprisingly, the scheme representation might influence the performance of chunkers when trained to produce these labels. For instance, [Sang and Veenstra 1999] compare the performance of seven schemes for the same task and [Ratinov and Roth 2009] reports the advantages of the BILOU scheme over the BIO for name entity recognition (NER).

2.4 (MONOTONE) AUTOMATIC TRANSLATION

Automatic translation systems from one natural language to another typically proceed in three stages: analysis of the source or origin text to produce a representation, transformation of this representation to make it nearer to the target language and, finally, a generation stage to obtain the translated text given the transformed representation. Translation systems are usually classified as direct, transfer-based or interlingua, depending on the level of interpretation performed during the analysis process, as illustrated by the well known *triangle of Vauquois* [Vauquois and Boitet 1985] shown in Figure 6.

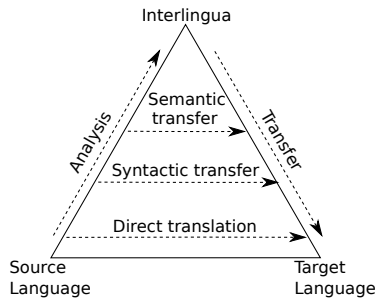


Figure 6: Triangle of Vauquois.

3 Chunks are non-exhaustive, meaning that some words may not belong to any chunk.

In this work we are interested in "sequence labeling" problems which can be modeled by statistical methods estimated from data, so we will limit this brief review to describe some type of statistical machine translation (SMT) systems which can be related, at least *partially*, with this task. For a more complete review of SMT, we refer the reader to some papers and surveys such as [Lopez 2008].

SMT systems [Brown *et al.* 1993] are based on the use of a large set of previously translated text,[4] known as a *parallel corpus*, in order to train models to estimate the conditional probability $p(t \mid s)$ that a translator will produce t, when presented with the source text s. Since there may be several possible translations, the task of SMT systems consist in finding the most likely one. One of the most successful type of SMT systems directly model this posterior probability [Och and Ney 2004]. Nevertheless, by using Bayes' theorem, this tasks may also consist in finding $\operatorname{argmax}_t p(t)p(s|t)$ where $p(t)$ represent the a priori probability of text t in the target language, whereas $p(s|t)$, known as *translation model*, estimates how well t expresses the same meaning as s. Note that the task consisting in modeling $p(t)$, known as *language modeling* (LM, studied in Chapter 6), does not require a parallel corpus.

It is well known that, in natural languages, certain contiguous word sequences form idioms whose semantics do not depend on the meaning of words which they are made of, so it is interesting to translate them as a unit.[5] This is the idea underlying the use of phrases, sequences of consecutive words, as the translation unit in transfer or direct SMT systems.[6] The set of phrases and their corresponding translations is obtained by means of the automatic alignment of a parallel corpus. Note that the use of phrases avoids the need of some artifacts such as the concept of "fertility" which is used in word based SMT systems to explain why some words in one language may lead to several words in the corresponding translation.

sequence labeling does not consider reordering

Translation is, in general, a problem much more complex than sequence labeling since translation units with equivalent meanings do not always appear in the same order in both sentences, making necessary to consider re-orderings in the translation process. SMT systems can be characterized by the amount of re-ordering considered during the process. In particular, monotone SMTs do not perform re-ordering but may be used, nevertheless, to successfully translate between some pairs of languages with a similar syntactic structure. The use of phrases in monotone systems can tackle some local re-orderings.

Other translation units besides phrases have also been proposed. A generalized version of phrases, known as alignment templates, use word classes instead of the words themselves and may also include

4 Some works, such as [Ravi and Knight 2011], try to address this problem without parallel corpora.

5 Also, some collocations (word sequences composed by terms that co-occur more often than expected by chance) are preferred to other linguistically correct and understandable expressions. Collocations can also be handled by means of language models which contribute to score competing target sentences.

6 The frontier between direct and transfer systems seems a little fuzzy. For instance, some SMT systems categorize words, can this categorization and the reverse expansion considered (simple) analysis and generation stages? The task of identifying these word sequences is related to segmentation and, since their meaning may also depend on the context, some type of joint segmentation and disambiguation can be envisaged.

word alignment information [Och and Ney 2004]. Bilingual units are pairs of phrases, one from the source and other from the target languages. Estimation of sequences of bilingual pairs is just a case of language modeling. These LMs can be converted into transducers [Casacuberta and Vidal 2004]. In a similar way, a particular case of bilingual units known as *tuples* can be also obtained by means of a monotonous segmentation of parallel corpus with additional restrictions in order to obtain one segmentation for a given sentence pair [Crego *et al.* 2004].

Syntactic or hierarchical phrases [Chiang 2007] are another generalization of phrases which allow the presence of gaps inside them. There is a bijection between source and target gaps, and these gaps may be filled with other phrases recursively, hence the name *hierarchical*. These systems are based on synchronous context-free grammars which are an extension of context free grammars not covered in this work.

Despite of the fact that SMT is a more general problem than the joint segmentation and classification, some SMT decoding algorithms are quite similar in essence with the decoders useful for our task at hand. Also, since it is also possible to integrate (usually, automatic speech) recognition and translation [Vidal 1997; Ney *et al.* 2000; Casacuberta *et al.* 2004] it is interesting to consider this task when dealing with the type of uses of decoding, the constraints on the output these systems can produce or the possibility of integrating both decoders.

To conclude this brief review, note that SMT is not limited to obtain the most probable translation of a given sentence. There are other interesting problems involved in this field. For instance, the alignment of parallel training corpora. Also, as part of most SMT systems, or to combine several systems, a set of target sentence candidates can be first obtained in order to re-rank later by using additional models. Other problem related to SMT is the evaluation of their performance in order to compare different systems, to choose the best configuration or as the training criterion. Finally, some extensions to the SMT problem can be envisaged. For instance, instead of translating from a sole source text to a target language, multi-source SMT take as input texts from different source languages. In general, the text may have to be translated into several target languages.

2.5 SPOKEN LANGUAGE UNDERSTANDING

The goal of spoken language understanding (SLU) consist in extracting the meaning from natural language speech. This task is not limited to the representation of the meaning conveyed in the spoken utterance but may entail performing some action. Besides the difficulty of interpreting spontaneous speech utterances, which may be ungrammatical, SLU has also to deal with the output of ASR systems whose performance is far from perfect in (as is usually the case) noisy or uncontrolled environments. Nevertheless, SLU has been successfully applied to several problems such as, but not limited to, spoken dialog systems, question answering or speech summarization [De Mori 2007; Jeong and Lee 2008]. A general study of these difficult tasks is

out of the scope of this work (for a more complete introduction we refer the reader to [De Mori 2007; and references therein]), but some approaches to SLU have stages or decoding sub-tasks more related to our work, not to mention the interest in the relationship with ASR.

Several representations for representing natural language semantics can be found in the SLU literature. For instance, formal logic has been considered adequate to represent and to reason about the knowledge of an application. "Frames" are data structures representing concepts. They contain a concept name and a set of roles represented by slot/-value pairs, where values may be other frames.

There are basically two SLU paradigms: rule-based and statistical, although some systems may combine both of them. Statistical SLU models estimate the (joint or conditional) probability of sequences of words and concepts from examples annotated with their corresponding semantics. Training sentences can be segmented into chunks which are associated a semantic constituent [Pieraccini and Levin 1992]. Thus, many statistical SLU systems start by finding the sequence of concepts which best explain the sequence of words obtained from ASR systems, which is usually known as *concept tagging*. They can apply the same techniques of text chunking or POS tagging. It is even more general to consider this stage as a case of (possibly monotone) SMT where natural language is translated into a formal semantic language [Pieraccini *et al.* 1993]. This stage can be more or less coupled with ASR decoding. Note that the ambiguity observed in the speech recognition stage can be delayed to this SLU component by representing the output of ASR systems by means of a probabilistic lattice of concepts, thus reducing the propagation of errors which is usually found in decoupled systems, although it is also possible to completely integrate the automatic speech recognition and the understanding stages, specially when both of them are based on finite state models.

concept tagging

This review has been focused on "concept tagging" which seems closer to the type of problems we are interested in. But let us remark that SLU is not limited to this task: a set of attribute name/values pairs is usually extracted from the sequence of concepts in order to produce a more structured and hierarchical frame representation by means of rule-based techniques, although stochastic approaches have also been proposed [Hahn *et al.* 2010].

2.6 AMBIGUOUS KEYBOARDS, PREDICTIVE TEXT

A keyboard is usually considered a deterministic and reliable input, excepting human mistakes. The recovery of these mistakes is an interesting subject per se, but here we will address entry methods where some kind of ambiguity at the keystroke level, not due to user mistakes, has to be disambiguated at a higher level. For instance, when the number of items to be chosen is much higher than the available number of input keys, the output sequence is not a simply concatenation of keystroke labels.

In general, entry methods for ambiguous keyboard are predictive since it is convenient to provide feedback after each keystroke. This as-

pect differs from more classical sequence labeling problem where the entire input is at the system's disposal before the output is obtained. We will see that ambiguous keyboard systems have some resemblances with other problems such as speech recognition and, although the feedback provided by some of those methods is not exactly the same, they are even more similar to the interactive transcription paradigm.

Stenography

The stenotype machine is a chording keyboard machine used by court reporters to perform real-time verbatim reporting of speech. [Karat *et al.* 1999] reports a rate for transcription using a computer keyboard of 33 words per minute (wpm) in average, 40 wpm is considered fast and average professional typists can reach 50 to 70 wpm. Compared with that, a stenotypist can easily reach[7] write speeds between approximately 180 and 225 wpm at very high accuracy. Although the use of stenotypes is not as widespread as other much more popular ambiguous keyboard methods used today (such as those related with smartphones and similar devices), this example is interesting not only because it is related to other problems illustrated in this chapter, but also because it is, historically, one of the first examples of ambiguous keyboard entry methods.[8]

Figure 7: Stenotype and keyboard layout.

The operation of chording keyboards is based on pressing several keys simultaneously (which may resemble playing chords on a piano, hence the term "chording"): some keys associated to the left hand represent phone sets used to represent the initial consonants of a syllable, other keys (right hand) represent the final consonants and, finally, thumbs produce the vowels. In this way, a single keystroke can represent a "phonographic syllable", i.e. "consonant(s)–vowel–consonant(s)". Polysyllabic words will often require more than one stroke but the reverse also holds: a single stroke may translate more than one word.

The fact that a stenotype is chorded is not as important for us as the presence of ambiguities to obtain the orthographic words. According to [Smith 1973], it normally takes three times the length of time to manually translate and type the notes as it does to take them, so modern systems automates this process. This translation problem is quite similar to ASR since they both have to solve *homophone disambiguation*, word boundary identification, the presence of words out of vocabulary or the need to perform user adaptation (since different typists may use different short-hands styles). These tasks can be tackled with the same statistical techniques. For instance, user mistakes could be addressed

homophone disambiguation

7 According to `http://en.wikipedia.org/wiki/Stenotype`

8 According to Wikipedia, chording keyboards can be traced back to the "five-needle" telegraph station designed by Wheatstone and Cooke in 1836. The stenotype machine, on the other hand, was invented in 1868.

by means of error correcting techniques. The paradigm of interactive pattern recognition (Section 3.5.3) perfectly fits the problem of deferred steno translation, whereas entry methods for smart devices are closer to open captioning or real-time stenography.

Chinese input methods

This section does not try to be exhaustive nor complete, just to show the interest on using ambiguous input methods in some tasks. Chinese easily fits under the condition of having a number of items much higher than the available input keys, but note that besides keyboard-based methods, it is possible to enter Chinese characters by using handwriting, speech or other techniques not discussed here.

Modern keyboard input methods for Chinese input can be roughly classified into two main categories:

PHONETIC-BASED where the user types pronunciations which are converted into characters. The case of homophones is solved by asking the user to choose from a list of candidates. But pronunciation can differ among different speaking languages, which poses a problem. An example if this input method is Pinyin.[9]

SHAPE-BASED are based on the structure of symbols rather than on their pronunciation, making it possible to input characters even if we do not know how to pronounce them. For an example of this method we refer the reader to the Wubi method which is claimed to be also extremely efficient: every character can be written with at most 4 keystrokes and, in practice, most characters can be written with fewer.[10]

Predictive text

T9 is a representative example of the use of ambiguous keyboards on handheld devices with a limited number of keys. The acronym T9 stands for "Text on 9 keys" and is a patented technology for mobile phone keyboards which usually had a 9 key layout. Given a key sequence, a lexicon for a given language is used to obtain the list of compatible words. For instance, the key sequence `4663` could be disambiguated as the words "good", "home", "gone", "hoof", "hood" and so on [Davis 1991; MacKenzie *et al.* 2001]. In order to obtain the sequences of candidate words, T9 makes use of the a priori probability of occurrence of each word (unigram language model, see Section 6.6.1) in order to propose first the most probable ones.

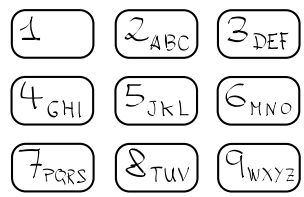


Figure 8: T9 layout.

This method is able to reduce, in average, the number of key presses[11] with respect to more straightforward multi-tap entry methods.

9 http://en.wikipedia.org/wiki/Pinyin

10 http://en.wikipedia.org/wiki/Chinese_input_methods_for_computers

11 This measure of performance will be reviewed into more detail in Section 3.5.3 associated to the more general interactive transcription paradigm.

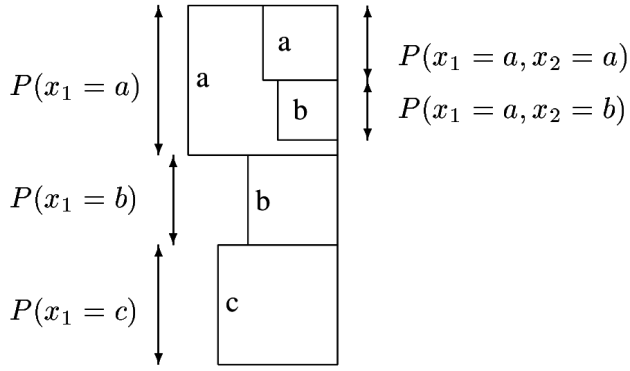


Figure 9: Suffix continuations are displayed in Dasher by means of rectangles whose area is proportional to the probability of those suffixes (Figure taken from [Ward *et al.* 2000; Figure 2]).

More sophisticated methods for predictive text input take into account not only the probability of isolated words, but also by considering the prior context of the word being typed. These types of entry methods are gaining popularity with the success of smart devices. Let us finish this section with two predictive text methods which are not based on ambiguous keyboards:

STANDARD KEYBOARD WITH PREDICTIVE TEXT ENTRY such as the interactive text generation systems described in [Rodríguez *et al.* 2010] where a text editor using conventional keyboards implements the strategy proposed by [Oncina 2009] to reduce, in average, the number of user corrections by suggesting the text which could be inserted after the current text being typed.

METHODS BASED ON CONTINUOUS GESTURES instead of the discrete events registered by conventional keyboards, could take more profit of the motor capabilities of users and can be more tolerant to inaccuracies. A nice and representative example of this approach is "Dasher" [Ward *et al.* 2000] which uses a continuous two-dimensional gesture input entry method (an eye-tracker, a mouse, ...) to select a continuously zoomable area of the screen divided into regions whose area is proportional to the probability of the prefix associated to it (see Figure 9). A quite popular text entry method for smartphones is illustrated in Figure 10.

Figure 10: Swype text entry method (image from `www.swype.com`).

2.7 HANDWRITTEN TEXT RECOGNITION

Handwritten text recognition (HTR) can be considered as part of a broader field known as "document recognition" which also covers layout analysis and is not limited to handwritten but also to typewritten or mixed handwritten-typewritten documents.

Since we are mainly interested in the relationship with the problem of joint segmentation and classification on sequences, our focus will be centered in the recognition of lines which, as described in the introduction of this work, can be considered as "fundamentally one-dimensional" signals[12] once they have been chopped into small pieces in the order of the lecture (remember Figure 1 borrowed from [Plötz and Fink 2009]). Moreover, since typewritten characters do not usually pose segmentation problems, we will deal with handwriting.[13] That is, although many of the techniques which can be applied to handwriting can also be applied to typewritten documents, it could be considered overkill in some situations.

Despite having narrowed our review down to handwriting, we will not be exhaustive and we refer the interested reader to [Plamondon and Srihari 2000] and their references for a more thorough exposition on the subject. Also, since Chapter 14 is devoted to present some novel preprocessing techniques relating to the normalization of handwritten lines, we will try to avoid to be redundant and avoid studying in depth many concepts which will be covered later into more detail.

The field of handwritten line recognition is usually distinguished by their input data modality into:

OFF-LINE when the image of the handwritten text is available at our disposal. This is the most general case since it suffices to scan the document previously written on paper.

ON-LINE when the text is written with a special device which allows us to capture the trajectory of the pen, as is the case of some special touchscreens. In this case, more information is available here than in the off-line case. Not surprisingly, on-line modality is usually considered easier. Indeed, on-line data can be converted into off-line in a straightforward way.

Handwritten can also be classified from another point of view depending on the difficulty levels, from easier to harder to recognize (see Figure 11), as follows:

- isolated characters, which may be tackled by classification techniques using a fixed set of parameters;
- words composed by isolated characters placed inside boxes, this task can be based on the previous one but we can take advantage of a lexicon to combine the classification of individual characters. This is an example of a joint classification problem,

12 This kind of approximation is not so uncommon. They are even done by some theoretical physicists which ignore some collapsed extra dimensions of our apparent reality to obtain effective theories.

13 Let us remember that the Sayre's paradox [Sayre 1973] described in the introductory chapter, was first related to handwriting.

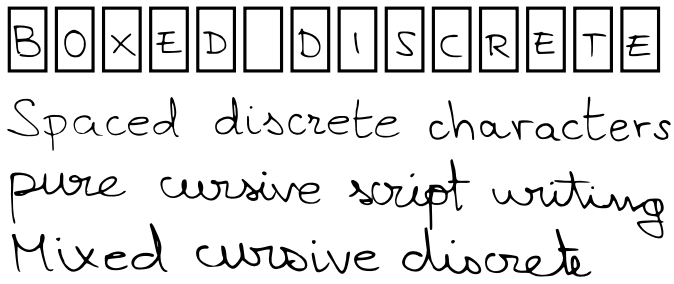


Figure 11: Difficulty levels on HTR of Western languages, from easier (top) to harder (bottom) (figure inspired by [Tappert *et al.* 1990; Fig. 2]).

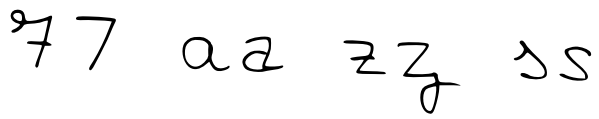


Figure 12: Examples of allographs (variants of the same grapheme).

- words composed by spaced discrete (isolated) characters is somewhat easier to solve than a pure cursive script writing and can be found in some online recognition systems;
- continuous handwriting may consider the previous problem as a particular case since we can mix spaced discrete characters and pure cursive script writing in the same line.

From now on, when talking about HTR, we will consider the most general case of continuous handwriting which can be considered quite similar to ASR: words are usually made of phonemes in ASR and made of graphemes in HTR. We can find co-articulation effects in ASR and a similar dependency between the realization of adjacent units is also found in HTR. In the same way that different realizations or variations of phones are called allophones, we can talk about "allographs": variants associated to the same grapheme, as illustrated in Figure 12. Also, both approaches can use a parametrization stage which produces a sequence of feature vectors also known as frames. Indeed, once this frame sequence has been obtained, the rest of the decoding process can use the same type of models and decoders. It is very important to take profit of similarities between ASR and HTR from the decoding point of view, but also to profit some differences which can be exploited by decoders: One of these differences is the possibility of obtaining an external over-segmentation of text lines into words. This process seems quite reliable in handwriting since it is difficult to find touching written words in the same line. This is not the case of continuous ASR when continuous words can be pronounced without pauses.

Let us finish this section with some common preprocessing stages, they will be covered into more detail in Chapter 14. We also want to use HTR as an example to show that the output of a preprocessing stage for problems initially considered related to sequences can be more general. More general representations such as directed acyclic graphs can be devised to deal with the "pipeline problem" as described at the end of this chapter.

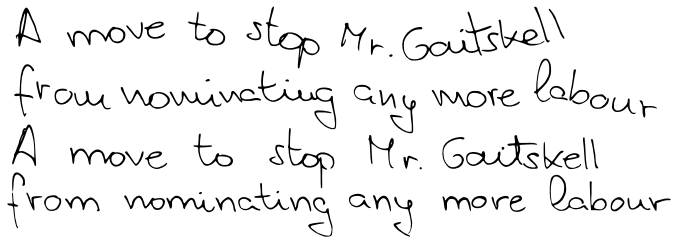


Figure 13: Illustration of slope: the slope of upper paragraph is much more pronounced than the slope in the lower one.

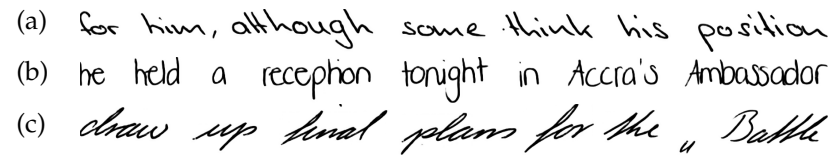
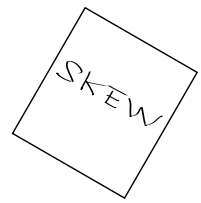


Figure 14: Examples of different slant angles (figure similar to Figure 189): (a) text slanted to the left, (b) without slant, (c) to the right.

Handwritten preprocessing stages

Off-line HTR is usually a part of document recognition which usually starts by scanning a sequence of pages. A common stage known as image cleaning or enhancement is usually applied in a first place to remove noise and to normalize the gray level. As will be explained in Chapter 14, some systems also perform image binarization, but others (as is our case) prefer to normalize the gray levels and use intermediary values as a useful way to model background/foreground ambiguities.

Another stage is the normalization of the skew of the pages. Skew is the overall rotation of the page which is usually due to the placement of the page in the scanner, either by hand or by automatic document feeders. This skew orientation reduces the accuracy of subsequent stages and that it is corrected. There are many methods to estimate the skew, some of then rely on projection profile methods, others on the Hough transform.



A general document recognition system has to deal with arbitrarily complex layouts which comprise not only text line blocks but also figures. These systems can work in a bottom-up, in a top-down or in other maybe mixed ways. In any case, lines usually belong to text blocks and have to be extracted from them. Text line extraction can also be based on histogram projection profiles, although more complex text blocks cannot be segmentation so easily and the overall system could benefit if this process could provide several solutions in order to model this ambiguity in order to delay it to following stages.

Once a line has been extracted, handwritten blocks are rotated and vertically displaced such that the lower baseline is aligned to the horizontal axis of the image, this process is known as slope correction and is depicted in Figure 13.

slope

Slant is the clockwise angle between the vertical direction and the direction of the vertical text strokes (see Figure 14). Slant correction transforms the word into an upright position making the image independent of this factor. This normalization is crucial if we adopt the

slant

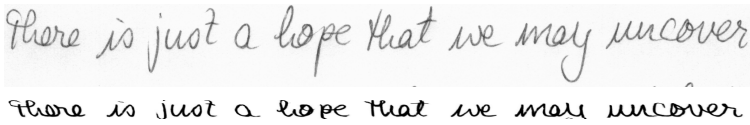


Figure 15: Example of text line normalization using the methods proposed in Chapter 14. The upper image corresponds to the first line of image from Figure 16, at the bottom we can see the normalized line.

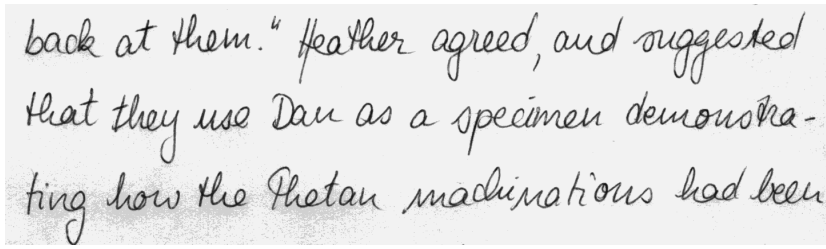


Figure 16: Example from IAM database. We can observe that *"demonstrating"* is broken at the end of a line *"demonstra-"* + *"-ting"*. Figure 17 shows a possible way to deal with these kind of situations.

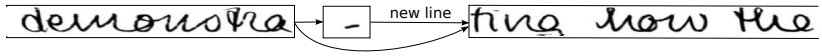


Figure 17: Example of a method proposed to deal with the possibility of hyphens at the end of line. Using the example from Figure 16 (line finishing with *"demonstra-"*) we can observe two different possibilities: a) either the hyphen breaks a word between consecutive lines, or b) the glyph which resembles an hyphen makes part of the end of the line and, since there is no broken word, a new line has to be taken into account by the decoder.

size normalization

idea of chopping the text line into image columns to obtain a sequence of feature vectors. Finally, size normalization tries to make the system invariant to the character size and to reduce the empty background areas caused by the ascenders and descenders of some letters, as illustrated in Figure 15. Classical approaches rely on geometrical heuristics to solve most of these stages. Some novel techniques to correct the slope, the slant and to normalize image lines based on automatic learning techniques are proposed in Chapter 14.

Handwritten words which have no place at the end of a line may be divided at the nearest break-point between syllables instead of moving them without breaks to the next line. A hyphen is inserted to indicate the presence of a word fragment, as can be observed in Figure 16. HTR systems ignoring that produce sure errors, others could apply ad hoc techniques to detect the presence of a hyphen at the end of the line although, since this hyphen could also appear for other reasons, there is a problem of ambiguity. We believe that this problem and other ambiguities during text line segmentation and normalization cannot be completely solved at the preprocessing stage. Thus, we advocate delaying hard decisions to subsequent stages by using mechanisms to efficiently represent a set of preprocessing alternatives as illustrated in Figure 17 for the problem of hyphens at the end of lines. These ideas will be discussed in Section 2.9.3.

2.8 SPEECH

Speech is one of the most ancestral and perhaps the most natural and ubiquitous way of human communication. Speech communication should be considered intrinsically bimodal [Chibelushi *et al.* 2002] since it takes into account the body language, gestures, facial expressions and other visual feedback between speakers. Nevertheless, in many practical situations, the visual information is not available so, for the sake of simplicity, we do not consider the visual modality in this review. There are many speech related tasks usually grouped under the cover term *voice recognition*:

SPEECH RECOGNITION (abbreviated ASR), to obtain the textual transcription of what has been said. ASR systems are usually classified using many different criteria: speaker-dependent or speaker independent, isolated or continuous word recognition, domain dependent or domain independent. There also exists many applications such as dictation, among others, which are also termed by the size of their lexicon into:

SMALL VOCABULARY up to 1K words;

MEDIUM VOCABULARY up to 10K words;

LARGE VOCABULARY more than 10K words;

VERY LARGE VOCABULARY more than 100K words.

WORD SPOTTING to determine the presence of a given word in a speech signal, ignoring the presence of other words;

SPOKEN LANGUAGE UNDERSTANDING (reviewed in a previous section) to extract the meaning;

SPEAKER RECOGNITION to guess who is speaking, although there are similar tasks such as *speaker verification* (authentication) or *speaker diarisation* (partitioning an audio recording into segments according to the speaker identity).

speech synthesis

Besides recognition, we can also mention *speech synthesis* which aims at producing an speech signal as natural as possible from a text transcription. Some speech synthesis systems are based on the same type of statistical models used by ASR [Zen *et al.* 2009]. Anyway, speech synthesis requires to take into account prosodic features and more fine grained phonetic details than those usually required by recognition.

ASR systems are fragile

This section is focused on ASR, which has been traditionally considered a very difficult task. This might seem surprising, at a first glance, given how easy seems to most of us. After more than 60 years or research, ASR has made an amazing progress[14] and may now be used in many practical situations, but ASR systems are far from human performance, they are very "fragile", meaning that their performance dramatically degrades with the presence of ambient noise, the use of spontaneous speech (plenty of disfluencies, filled pauses, repetitions, repairs,...) or when several speakers talk simultaneously (e.g. like in

14 Indeed, the term "super-human" can be found in some research papers to denote the fact that their systems outperforms human listeners.

a "cocktail party"). One of the reasons of this difficulty is the presence of a great variability in speech. Not only inter-speaker but also intra-speaker variability, since the same person produces different speech signals when pronouncing the same words. The speech signal contains not only linguistic, but also many other types of para-linguistic information such as intonation, rhythm, speed, tone and other clues about the emotional state, as well as indications about the speaker: her/his gender, age and even the cultural or regional origin (not only given by the accent, but also by the use of some words or expressions).

quintessence of sequence labeling

The great variability and complexity of speech, together with the fact that ASR has been historically one of the first tasks where some statistical techniques (now applied to many other tasks[15] described in this chapter) were employed explains why it has been traditionally considered the "quintessence" of sequence labeling problems and of the joint segmentation and classification problem on one-dimensional signals: despite this complexity, speech is made up of a set of segmental sounds (phones) whose different realizations are grouped into a reduced set of abstract classes (phonemes, which are particular to each language) which may be realized differently (allophones) depending on the context (among other phenomena, this is explained by co-articulation). Phoneme sequences are grouped into words following a lexicon (and homophone words share the same phoneme sequence) and words sequences are modeled by language models.

Let us try to provide a little background on speech tasks to better understand or to better put into context some of the methods described later. Note that we do not aim here at explaining how ASR systems work nor to provide an historical perspective or a complete bibliography, but just to show where its difficulty comes from and to understand how some models studied in the following chapters can be applied here. As with other sections, we refer the reader to the extensive bibliography such as [Jurafsky *et al.* 2000; Chapter 7] or surveys such as [Benzeghiba *et al.* 2007] for a more detailed information on speech variability.

Speech production

Although researchers usually remember the saying *"an aircraft do not need to flap its wings in order to fly"*, the study of speech production and perception is interesting to understand how the speech signal can be best represented to perform ASR,[16] to better understand how phonemes can be classified, to explain co-articulation phenomena and also to understand which part of the speech variability is due to speaker anatomic differences (such as the vocal tract length). The speech production can be roughly described as the following source/-filter model:

15 The adaptation of these techniques to other tasks, notably handwriting recognition, seems now trivial to us, but they were historically proposed as not straightforward adaptations from ASR.

16 Nevertheless, some methods exclusively based on pure statistical analysis of speech corpora, without any assumption about how speech production and perception really works, give support to the same results on how the speech signal is best represented.

SOURCE most speech sounds are produced by expelling air from the lungs. When air passes through the larynx, it traverses the space between the vocal folds (or vocal cords). The vocal cords together with the space between the folds is known as glottis. When the folds are tensed and close enough, they vibrate producing a regular wave with a fundamental frequency which depends on the anatomical characteristics of the speaker[17] producing sounds known as voiced, whereas sounds made without this vibration are called unvoiced.

FILTER refers to the vocal tract, which is composed of different resonating cavities (oral and nasal tracts) and articulators (tongue, jaw, velum, lips, mouth) situated between the glottis and the lips.

Phoneme classification

As explained before, a phoneme is an abstract class representing the smallest segmental unit of sound or group of different sounds perceived to have the same function to distinguish utterances by speakers of a particular language. The concept of allophone is used to denote different surface realizations (phones) of phonemes.

The source and filter speech production metaphor can be used to categorize phonemes by how they are produced. Moreover, since the characteristics of the articulatory machinery varies continuously over time, this also explains the co-articulation effects (whose influence can be even observed over several phonemes), and the influence of the rate of speech. Phones are classified by how they are produced into:

CONSONANTS are produced by blocking or restricting the airflow unvoiced. They can be distinguished by where this restriction is made in the vocal tract (known as the *place of articulation* and classified as labial, dental, alveolar, palatal, velar or glottal) and also by how this restriction is performed (known as *manner of articulation*). The combination of the place and the manner of articulation may be used to identify consonants and to further classify them, for the English language, in:

FRICATIVES f,v,s,z,sh,j;
STOP/PLOSIVES p,t,b,d,g,k;
NASALS n,m,ng;
SEMIVOWELS (LIQUIDS, GLIDES) w,r,l,y.

VOWELS have less obstruction than consonants, they are usually voiced and their duration may be longer. They can be characterized by the position of the articulators (mainly the height of the tongue and shape of the lips) and this has an acoustical counterpart in the location of formants (the resonant frequencies) in the spectral representation of the speech signal. The smooth transition between different vowels are known as diphthongs and triphthongs (transition between, respectively, two and three vowels).

17 Mostly due to the vocal tract length, hence the use of vocal tract normalization methods. Also, its correlation with the gender explains the existence of male and female specialized acoustic models.

Sub-word Units

There is a trade-off when choosing a set of units to develop ASR systems. On the one side, we have seen that phonemes are abstract classes representing the smallest segmental unit of sound or group of different sounds perceived to have the same function by speakers of a particular language in order to distinguish utterances. So, at a first glance, phonemes seems the natural choice to model sub-word units. But, on the other side, the actual realization of phonemes when spoken (phones) depend on the neighboring phonetic context, and we would also like to tackle this variability. So, why not simply model words as the basic acoustic unit? After all, most co-articulations appear within words and we do not need to obtain the phonetic transcriptions of the lexicon. The problem is that, unless we were dealing with very low vocabulary isolated ASR systems, the number of acoustic models, of corresponding parameters and of training examples required to accurately estimate these models become too huge due to the *curse of dimensionality* (addressed in following chapters) not to mention the complexity of the resulting decoders. That is the reason why some "sub-word" units are normally envisaged.

When using sub-word units, a transcription of the lexicon in terms of these units is required, problem usually known as "grapheme-to-phoneme conversion". Some languages allow the use of phonetic transcription rules given by hand, in others we have to rely on dictionaries. Multiple transcriptions are often considered. Some authors remark that this problem is similar to SMT and apply automatic learning techniques [Bisani and Ney 2002].

function words

Some decoders use word and sub-word acoustic models at the same time. For instance, many *function words* (term usually associated to very frequent and short words) are modeled as special acoustic units to improve the recognition accuracy [Bahl *et al.* 1989b] while the rest of words are made of sub-words units.

A common way to alleviate the problems of co-articulations consists in using phonemes but distinguishing them by their neighboring context, leading to context dependent units [Lee 1990], known in general as polyphones: a biphone is a phone that depends only on one context, the left side phoneme or the right side one. Triphones take into account both contexts, quinphones take into account the two previous and following phonemes, and so on.[18] Since the number of context-dependent models is much larger than the original sub-word repertoire and since the number of training samples is also reduced, some clustering techniques may be applied to limit the number of different models, they can use a reduced context when there is no enough data to train the original model (using back-off mechanisms or other smoothing techniques). It is also possible to tie some parts of the models to reduce the number of parameters.

Other alternative to model transitions between phones consists in dividing the phone into two parts, known as "demiphone" [Mariño *et al.* 1997; Menéndez-Pidal *et al.* 2007]: the first part phone is mainly affected by the left side co-articulation and the second part is affected

18 And the term "monophone" is sometimes used to refer to context-independent units.

by the right context. Although the dependence of both side contexts is not taken into account, the number of models is reduced with respect to triphones. Stationary-Transitional Units [Fissore *et al.* 1996; Gemello *et al.* 1997] are based on similar ideas.

Some researchers distinguish different time scales in speech: sub-segmental features can be analyzed in a range between 25 and 80ms, whereas the phonetic information is integrated at syllabic timescales (150–300ms). That is the reason why the use of syllable sized units has also been propose for phonetic decoding [Wu *et al.* 1998; Ganapathiraju *et al.* 2001; Jansen and Niyogi 2009]. We can also find "Demi syllables", where syllables are cut into onset and rhyme, leaving the vocalic part to the latter [Schukat-Talamazzini *et al.* 1992].

Sub-word units associated to phonemes and syllables are based on concepts given by phoneticians, but it is also possible to envisage a set of units determined automatically from a corpus of speech utterances, as is the case of fenones and multones (a set of fenones in parallel) which are obtained by means of clustering and vector quantization [Bahl *et al.* 1991]. A similar automatic clustering of HMM states with loops leads to a representation known as senones [Hwang and Huang 1992]. Senones are sub-phone which constitute a particular way to implement acoustic models sharing common parts. For instance, triphones may be made from senones so that the same senone may appear in different triphones. The reasons behind this sharing is threefold: to improve estimation of parameters, to reduce the memory footprint and the computational cost.

Speech signal representation

Any pattern recognition task requires a proper preprocessing and feature extraction in order to highlight the aspects required to classify while reducing the non relevant variability. The speech waveform captured by microphones is sampled for digital processing. Since the range of frequencies relevant to human speech is from 50 to 7000 Hz, a sampling frequency of 14 000 Hz usually suffices.[19]

The parametrization of speech signals is an active area of research. Most common feature extraction methods rely on the time-frequency decomposition of the speech signal. The source and filter model of speech production explains why the shape of the spectral envelope is relevant to classify speech sounds: speech varies over time according to the source signal and the configuration of the speech articulators.

Despite of the fact that speech is a non-stationary signal, it can be approximated by a stationary one in a window between 10 and 100 ms of duration. That is why most ASR front-ends analyze short signal windows called "frames" with a duration usually between 20 and 35 ms on which stationarity is assumed. The spectral energy associated to each frame are grouped into a finite set of frequency bands compressed in the frequency domain according to the sensitivity of humans to different frequencies. The Mel scale, obtained from speech perception studies,[20] is the most commonly used to this end.

19 Lower frequencies may be employed due to limitations of transmission channels.

20 Frequencies expressed in Mel scale are equally spaced according to human perception.

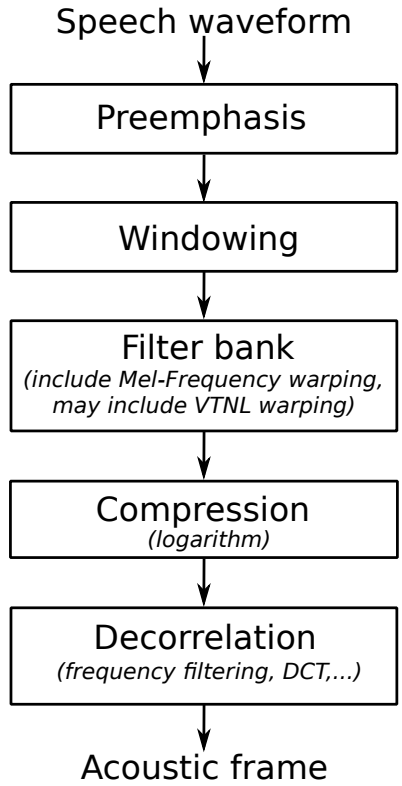


Figure 18: ASR front-end.

A bank of filters is usually computed using the energy obtained by the fast Fourier transform (FFT) and the energy associated to each band is usually compressed using a logarithmic function (see Figure 18) also to mimic perception issues with the additional benefit of converting the variations of gain due to the microphone into an additive factor which can be easily removed. Finally, some type of de-correlation and parameter reduction is applied to these values, Mel Frequency Cepstral Coefficients (MFCCs) relies on the Discrete Cosine Transform to this end. Other filter bank spacing besides the Mel scale are possible, such as the Bark scale or the uniform scale. Indeed, it is not clear that one of them outperforms the others [Shannon and Paliwal 2003]. Besides the discrete cosine transform, other type of de-correlation is the "Frequency Filter" parametrization proposed in [Nadeu *et al.* 1996], which is the technique chosen for our own ASR experiments. A common practice consists in normalizing these features by subtracting the mean and by dividing by the standard deviation.

We can observe that, if the rate at which frames are sampled is high enough to preserve the properties of speech that convey the message, no information will be lost. That is why frames are overlapped in order to obtain a new frame every (typically) 10 ms. Also, since dynamic features are important for speech perception, many front ends include delta features (first and second derivatives).

Neighboring frames are very correlated for several reasons: windows are very overlapped, they measure configurations whose change depend on mechanical components (the same that also explained coarticulation) not to mention the properties of time-frequency representations which implies that high temporal resolution and frequency resolution cannot be achieved at the same time.[21]

Despite the existence of more sophisticated time-frequency representations (for instance, based on multi-resolution analysis), the most widespread feature extraction methods for ASR (such as MFCCs) rely on frame windowing and stationarity assumptions.

21 Known as the Heisenberg–Gabor limit, to achieve a good frequency resolution a wider window is required at the cost of losing temporal resolution, while a narrow window has the opposite trade-off.

Some other issues to take into account

Let us finish this partial review of the ASR field with a brief description of additional issues or features of speech which can be taken into account by ASR systems:

RATE OF SPEECH has an impact on both temporal and spectral characteristics of the signal, so it is often desirable to estimate the rate of speech and to take this information into account in different ways, for instance: the use of specialized acoustic models or to adapt the decoding process.

There are several ways to estimate this rate without performing ASR, such as the use of signal-processing metrics [Morgan and Fosler-Lussier 1998];

EXTERNAL OVER-SEGMENTATION is the idea of providing some information about the division a signal into phones, words, etc. without relying on ASR, as an aid to improve decoding performance. ASR systems *do not rely* on this kind of information and prefer to use techniques known as "segmentation free".

Nevertheless, some segment based approaches use a segmentation stage which can be computed by means of spectral measures or with the aid of a first cheaper recognition stage;

VOCAL TRACT NORMALIZATION consists in applying a frequency warping to reduce the mismatch between the characteristics of the speaker and the characteristics observed in the training samples with respect to the length of the vocal tract, which is an important source of speaker variability [Andreou *et al.* 1994; Lee and Rose 1998].

The optimum warping factor is usually chosen to maximize the average likelihood of a sequence of observations with respect to a model and this warping factor is applied to the array of filter bank coefficients. [Miguel *et al.* 2008] proposes a technique to allow local adaptations of the vocal tract factor by using a set of warping functions and by allowing the decoder to search in an expanded space which takes into account the possibility of using any of these warpings while imposing some inertia (adjacent frames have to use adjacent warpings). An alternative, more general approach, will be discussed in Section 2.9.3;

WORD-BOUNDARY EFFECTS means that besides the co-articulation observed between neighboring words, there may appear other phenomena such as the fusion or alteration of sounds across word boundaries, which is known as "external sandhi",[22] "Liaison" (in French) or "raddoppiamento fonosintattico" (in Italian). These phenomena can be taken into account by decoders which use context-dependent units [Alleva *et al.* 1992], which can be classified either as "intra-word" or as and "across-word" context dependent decoders.

22 Sandhi (`http://en.wikipedia.org/wiki/Sandhi`) is a cover term for a wide variety of phonological processes including the fusion of sounds across word boundaries and the alteration of sounds due to neighboring sounds or due to the grammatical function of adjacent words.

PHONOLOGICAL RULES in general can also be tackled by decoders since this information can be expressed by using finite state methods [Kaplan and Kay 1994; Hetherington 2001] and can be easily and elegantly integrated in some ASR systems based on composition of transducers;

TONALITY is the use of pitch to distinguish lexical or grammatical meaning in some languages. For instance, in Mandarin Chinese each syllable carry their own tone from a repertoire of 5 different tones which leads to different lexical meanings. ASR can take into account the tone by increasing the repertoire of sub-word units [Chen *et al.* 1997], although some systems prefer to use a multistage approach where an imperfect model is used in a first stage to obtain syllable boundaries and the tone differences are determined in a second stage [Zhou *et al.* 2004];

PHONE DURATION plays a significant role in the comprehension of some languages such as Finnish where there exist many word pairs which are distinguishable mainly by the duration of their phones [Hirsimäki *et al.* 2006]. For instance, the following six Finish words, taken from [Pylkkönen 2004], differ only on the durations of their phones: *taka* (back-), *takaa* (from behind), *takka* (fireplace), *takkaa* (an inflection of *takka*), *taakka* (load) and *taakkaa* (an inflection of *taakka*). The explicit control of duration will be discussed (not limited to speech) in Chapters 7 on segment modeling, and it will be clear that taking explicitly into account the duration of realization of sub-word units requires more computationally expensive decoders. This, together with tonality, are examples of the trade-off usually found between accuracy and efficiency in many pattern recognition tasks;

HYPER–ARTICULATION is related the use of speech for correcting words which have been misunderstood in a previous ASR iteration.[Yu 2008] expresses very well this concept:

> *Another common phenomenon with spoken corrections, particularly re-speaking, is hyper-articulation, a manner of speaking characterized by more pauses between words, longer pauses, and less disfluencies. This "clearer" type of speech is a person's reaction to another human misunderstanding what was said. While stronger enunciation may improve human understanding, it does the opposite for speech recognition systems. Standard acoustic models are not trained on this type of speech and thus recognition performance suffers, leading to higher error rates and cascading errors.*

Note that the list is not exhaustive and the ASR field requires to take into account many other issues. Specially relevant for the good performance of ASR systems are the use of speaker adaptation techniques or the availability of good speech activity detection and speaker diarisation systems, to mention a few. Note that dealing with possible mismatches between training and test data or conditions, are not particular of ASR systems but are generally applicable to any other problem.

2.9 SUMMARY AND SOME CONCLUSIONS

other problems

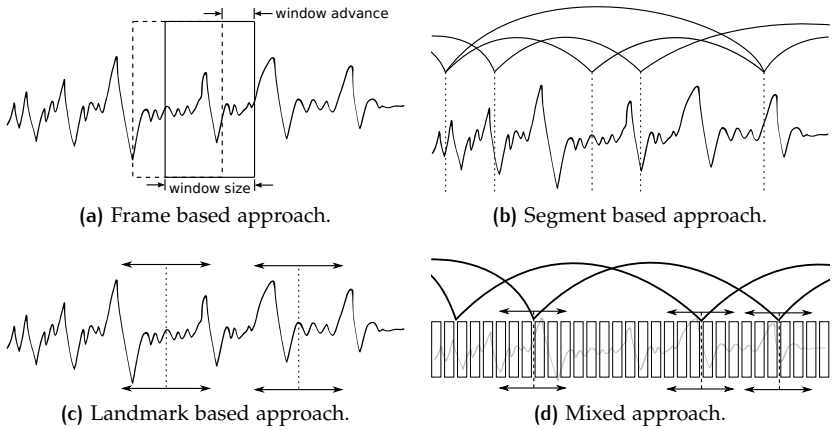
We have shown some problems which can be expressed, at least partially or for some particular cases, as problems on sequences. We have left aside other interesting problems which can also be solved by means of similar techniques: context sensitive spelling correction, expansion of "Grade 2 Braille contractions", sign recognition, audio-visual speech recognition, information coding, protein secondary structure prediction, analysis of electrocardiogram signals, technical analysis of financial data and so on. Some of the problems illustrated in this chapter, such as word division, only require to segment the observed sequence into contiguous parts. For other problems it suffices to classify the entire sequence into a predefined set of classes, or even to detect the presence of some feature in the observed sequence. The most general type of problem we will try do tackle requires to segment a sequence and to label/disambiguate each segment, being both tasks interrelated. Some problems only involve an observed sequence, but others require to establish a correspondence between two or more sequences (e.g. parallel corpus alignment, multi-stream ASR). We will try to list and to relate these types of problems into more detail in Chapter 3.

some limitations

Note that there also exists problems on sequences which are out of the scope of this work. For instance, general SMT involves re-orderings whereas a joint segmentation and classification naturally leads to associating a signal with a sequence of labels in a monotone way. We also saw that general SLU may use semantic frames which are hierarchical structures, so that the techniques discussed here could only cover the very first steps (e.g. concept tagging). The general document recognition field cannot be considered a problem on sequences because it involves layout analysis which may also produce a hierarchical or even more complex description. We can observe the interest of some problems apparently out of the scope of this work provided some particular sub-task can be accomplished. Also, most concepts discussed in this work, such as the Sayre's paradox, can be generalized to higher dimensional data and *some* decoders which take re-ordering into account are not so different to those who do not deal with it.

frame sequences

Previous examples described many different tasks which seem different on surface but can hopefully be posed under the same framework after a domain dependent preprocessing. A clear example of such tasks are ASR and HTR which, after producing a sequence of frames, can be tackled by means of the same techniques. We have also observed that, although in some problems the original sequence is a sequence of symbols, in others it is a continuous signal. Continuous signals have to be approximated by a sequence of feature vectors in order to use the same techniques as with the previous examples. This seems adequate as far as signals are sampled at a sufficient rate to preserve the information and the properties we try to analyze.



(a) Frame based approach.

(b) Segment based approach.

(c) Landmark based approach.

(d) Mixed approach.

Figure 19: Frame, segment and landmark based approaches: In **(a)** a feature vector is obtained from the contents of a possibly overlapped sliding window obtaining frames at a fixed rate. In **(b)** a finite dimensional feature vector is obtained for each segment, although other models also called "segmental" or "segment based" differ from this approach. In **(c)** a set of "landmarks" (positions in the signal) are selected and feature vectors are obtained for each landmark taking into account the surrounding signal. As depicted in **(d)**, it is possible to obtain segment and landmark information from frames and it is also possible to mix landmarks and segments in the same approach.

2.9.1 Underlying representation

alternatives to frame sequences

We have seen that, in general, the underlying representation in the problems described in this chapter basically consists of a frame sequence obtained at a fixed rate, but there exists other alternatives (depicted graphically in Figure 19. The list is not exhaustive, but covers three main approaches we have observed in the ASR literature, although they can be adapted to other tasks):

FRAME BASED where a frame sequence is obtained using a sliding window usually at a fixed rate. This is the feature extraction process assumed by default in most tasks described in this work when departing from a continuous signal. There exist some variations allowing the use of variable width sliding windows, to non fixed rate[23] or to take several frame streams into account;

SEGMENT BASED the speech signal is divided into segments (usually taking only a subset of all possible ones by means of some type of "external" segmenter) and a feature vector is obtained for each segment. Observe that segments are represented as a fixed set of features no matter the segment length. These type of models will be discussed into detail in Section 4.4.2. Let us also remark the existence of models called "segment models" based on this type of observations and other models using the same name which are based on frame-based observations;

23 For instance, the possibility of dropping frames to increase decoding performance [Duvinage and Parfait 2010].

LANDMARK BASED is based on the idea that a set of acoustic events or "phonetic landmarks" located at precise "anchor points" of the speech waveform (e.g. in consonant closures and releases) carry the most significant information.

To our knowledge, the landmark approach has only been proposed for ASR tasks and is essentially related to certain theories of human speech perception [Stevens 2002] so it seems to be completely particular for ASR tasks and cannot be adapted to other non-speech problems in a straightforward way. In this theory, the speech is represented by means of a sequence of segments, each one described by a set of distinctive (usually binary) features.

In this context, landmarks are used in a first stage followed by an analysis of the acoustic information extracted around each landmark in order to obtain the sequence of sets of features stated by the model. There may exist different types of landmark and the location of each type may depend on different features (e.g. spectral-change metrics, detection of peaks in some formants, etc.). Also, different types of features are extracted around each landmark position, possibly using different resolutions, depending on their type. Since only most relevant parts of the signal are analyzed in the landmark based approaches[24], they are claimed to adapt the level of effort where it is needed[25] [Espy-Wilson *et al.* 2007].

Although landmark locations may not neatly correspond to phonetic boundaries, many of them are associated to them, leading some researchers to use the terms "landmark" and "segment boundary" as essentially equivalents. There exist some works [Glass 2003; Ström *et al.* 1999] which identify "landmarks" with "segment boundaries" and extract features summarizing the spectral shape around them to fed statistical models, making them close to variable-frame-rate analysis and making it easier to combine landmarks with segments in the same statistical setting.[26] They seem also more suitable to be applied to other different tasks since they are not based on precise knowledge based approaches but on the more general idea of locating "anchor points" and extracting features around them.

It is quite remarkable the relationship between the models used in ASR and the phonological representations of the underlying linguistic and phonological theories: the phonological representations of most ASR systems consist of a phonemic sequence as described in the generative phonology[27] originated in the 1960s [Chomsky and Halle 1968]. On the other side, landmark approaches, based on perception-oriented theories, are more related to discriminative techniques.

---

24 The entire signal is analyzed, nevertheless, in order to find the landmark positions.

25 Although we have to remark, regarding this issue, that HMM parametrization represents a negligible part of the overall computation effort and, when HMM decoding uses pruning techniques, the effort is usually located in the most confusing parts and not limited to phonology but also including semantic issues.

26 After all, the same multiple segmentation stage is required in both cases, and a finite dimensional feature vector associated to each segment and to each segment boundary, which can be tackled in similar ways. This thread will be retaken in Section 4.4.2.

27 The fact that speech segments were classified, in this theory, by means of binary features (e.g. nasality, voicing,...) is mainly ignored in favor of a more general hypothesis: a finite repertoire of units.

2.9.2 (Some notes on) preprocessing

feature extraction

Although preprocessing and feature extraction stages are domain-specific, they share common aims and general techniques (dimensionality reduction, variable scaling, de-correlation...) to provide an adequate representation or encoding in order to simplify the following tasks. They depend not only on the domain but also on the techniques to solve them. For instance, some models are more sensitive to the relative scale of different input variables or may assume some type of distribution on the data, which can be achieved by means of de-correlation techniques. Among the de-correlation techniques based on linear transformations the most common ones are Principal Components Analysis (PCA) [Jolliffe and MyiLibrary 2002] and Linear Discriminant Analysis (LDA). Regarding non-linear techniques, both neural networks auto-encoders[28] and self-organizing maps are well known.

dimensionality reduction

These dimensionality reduction techniques can be applied to each frame in frame sequence representations or, generally, to any fixed dimension feature vector. But we have observed in previous section that other underlying representations are possible for problems on sequences, namely: segment based and landmark based observations. Let us briefly discuss some general methods to obtain a fixed dimension feature vector for a variable length sequence: trace segmentation [Castro and Casacuberta 1991] has been used in isolated speech recognition and consists in uniformly sampling the trajectory of the utterance in the frame space. In that case, stationary parts of the signal, where the information is redundant, are usually compressed because their distance along the trace is small, while transitional portions of speech corresponds to longer sequences over the trace. Other ways found in the literature to represent a variable length speech segment by means of fixed dimensional vectors split the segment into a fixed number of fragments in order to obtain the mean value of the frames of each fragment. It is also possible to use of the discrete cosine transform over a variable length signal [Harte *et al.* 1998] and even the computation of distances between the segment and a predefined set of prototypes [Fischer 2012]. A general way to represent variable length sequences via fixed dimensional vectors, given a generative model, is described in [Jaakkola and Haussler 1998] and is based on obtaining the gradient of the log-likelihood with respect to each parameter: this gradient describes how that parameter contributes to the process of generating the sample. Another promising approach consists in using convolutional networks as described in [Maas *et al.* 2012].

fixed dimension representation of variable length segments

2.9.3 Ambiguous front-ends and the pipeline problem

The underlying representations are associated to feature extraction processes which are followed by subsequent stages. The problem of this stacked, sequential or pipelined approach is that providing a unique feature representation implies taking a hard decision, so that

28 Neural networks with a hidden bottleneck layer which are trained to perform the identity function [Bishop 2006].

any mistake is propagated through following stages. These mistakes are very hard or even impossible to recover. This problem is known as the *pipeline problem* , or also the problem of cascading errors (see [Finkel *et al.* 2006]) and is well explained by [Felzenszwalb and McAllester 2007; Section 1.1]:

pipeline problem

> *A major problem in artificial intelligence is the integration of multiple processing stages to form a complete perceptual system. We call this the "pipeline problem". In general we have a concatenation of systems where each stage feeds information to the next. Because of computational constraints and the need to build modules with clean interfaces pipelines often make hard decisions at module boundaries.*

Although the authors of this quote propose a unique formalism based on weighted rules and lightest derivations, we share their main aim which consist in both enabling coarse high-level processing to guide low-level computation and avoiding making hard decisions between processing and following stages. [Finkel *et al.* 2006] describes different approaches to tackle the pipeline problem, among them, to pass k-best lists between processing stages or even to pass the entire distribution to the next level which, in turn, computes the full distribution to the following one.

In this work, we are going for the approach consisting in using DAGs as a useful generalization of sequences which can represent a large number of alternatives in a compact way while avoiding a combinatorial explosion during the process. Non recursive context free models (packed shared forests [Billot and Lang 1989] or, equivalently, acyclic recurrent transition networks (RTNs) as described in Section 5.4.3) are another interesting representation worth studying.

We hope that some examples, specially for HTR and ASR, show the interest in representing ambiguity during preprocessing to tackle ambiguity problems during text line extraction and other preprocessing stages and also to deal with words broken into lines. Delaying this non-determinism may affect the decoder as is the case of the ASR with the vocal tract normalization approach proposed by [Miguel *et al.* 2008] and discussed before: a preprocessing parameter (a set of warping factors) is explicitly taken into account during decoding. We believe that it is also possible to achieve the same results by including this source of variability (including the constraints over adjacent frames proposed by the authors) into the preprocessing stage so that it suffices, for a generic decoder, to be able to use a DAG instead of a sequence. Although this is an implementation issue, the more general stacked approach does not imply a lower efficiency since feedback mechanisms can be used to avoid generating an unnecessary larger search space as described in Chapter 8. The same idea of varying the vocal tract normalization factor in ASR can also be applied to HTR where, in this case, it is possible to perform a non-uniform slant correction. We propose in Chapter 14 a non-uniform slant correction algorithm based on dynamic programming which produces only one result, we believe that the possibility of obtaining a DAG varying slant angles is worth considering as a future work.

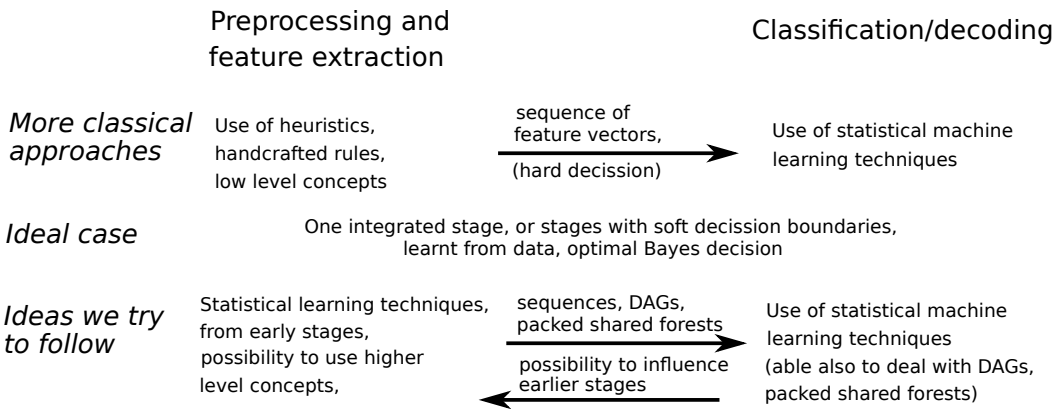


Figure 20: Some ideas relating preprocessing and feature extraction with regard to the pipeline problem.

We have graphically depicted some ideas relating preprocessing in Figure 20. Some of these ideas have been used or have inspired our work described in following chapters, they are summarized in the following list:

- to delay the conversion of soft estimations into hard decisions from the preprocessing to the following stages where more additional information sources or more task specific loss measures can be taken into account. To this end, use compact ways such as DAGs or non-recursive context free models (e.g. acyclic RTNs) to represent sets of alternative hypothesis;
- to bring automatic learning techniques to the very early stages of preprocessing, instead of relying on heuristics or handcrafted rules, as in the HTR preprocessing techniques proposed in Chapter 14;
- in a symmetric way, to bring some concepts of the latest stages forward. Some feature extraction techniques take into account concepts which do not belong to preprocessing but to the next stages. For instance, during HTR preprocessing we can talk about *ink blocks* but it has not sense to talk about *words* without a proper word segmentation stage. Nevertheless, some authors [Bharath and Madhvanath 2011] defines the aspect ratio of a stroke as the ratio between this stroke and the word where it appears. This measure makes sense in isolated word recognition, but it cannot be extended to continuous word recognition without taking into account the segmentation of strokes into words from the very beginning;
- both preprocessing/feature-extraction and decoding stages are interrelated despite the common independence assumptions. For instance, in ASR the speaking rate is correlated with the word variants employed by the speaker and, in general, the way we speak [Alleva *et al.* 1998; Morgan and Fosler-Lussier 1998; Fosler-Lussier and Morgan 1999]. Most ASR systems use a LM and

Figure 21: Document with some seeping ink from the reverse side of the paper (bleed-through effect). As can be observed, the handwritten text has a marked slant. This can be used as an additional evidence that some strokes with different direction correspond to background when the slant angle is assumed to be roughly the same in both sides.

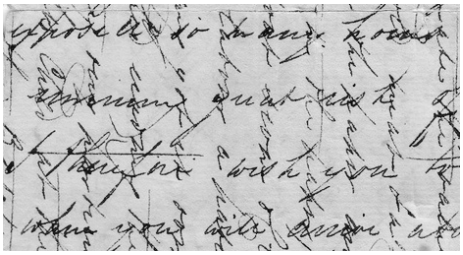


Figure 22: Fragment of a crossed letter where some text is written over the other perpendicularly (image source: wikipedia).

acoustic models as independent as possible to improve modularity which, in turn, means more easy to estimate parameters and better decoding efficiency, but [Ward *et al.* 2012] examine a number of prosodic and timing features as potential sources of information on what words the speaker is likely to say next;

- it is also possible to take the results of following stages as indicators to guide the previous ones. For example, the image denoising of scanned ancient documents is important for their recognition. The removal of seeping ink from the reverse side of the paper (known as the bleed-through effect) is easier when both sides of the paper are at our disposal. In that case, the presence of slant may help when the slant angle is roughly the same in both pages, as shown in Figure 21 where it can be observed that the presence of slant in the opposite direction gives an additional evidence that these strokes correspond to background. This example illustrates the dependency on stages which are usually performed later since slant removal is performed, in the usual HTR chain, after image denoising. Recognition of crossed letters (see Figure 22) might constitute another example of the possible dependency between recognition stages.

- This bidirectional communication may occur *while the preprocessing is being done and the data is being sent*. We have developed this idea in the form of a DAG serialization protocol. Using this protocol, the decoder can proceed as the data is being received while allowing it to send back optional information to the previous stage. This feedback information means that pipeline systems are not necessarily uncoupled as usually assumed.

3 SOME PROBLEM TYPES RELATED TO SEQUENCES

ONCE YOU LABEL ME YOU NEGATE ME.
Søren Kierkegaard

IN SCIENCE THERE IS ONLY PHYSICS;
ALL THE REST IS STAMP COLLECTING
Ernest Rutherford

CONTENTS

PREVIOUS chapter presented some examples of problems related to sequences. This chapter is focused on characterizing the *types of problems* associated to them. We do not pretend to establish a complete taxonomy although, or the sake of completeness, some problems on sequences which do not meet our assumptions are also described. Unfortunately, there is not a clear consensus on a precise terminology

for all these problem types.[1] Note that we are not dealing, for the moment, with the precise ways to solve these problems, we are just trying to describe them in terms of inputs and outputs, which is not always sufficient to clearly discriminate between some problem types.

In addition to the classical problems on sequences, some extensions to deal with multimodality or with collaborative and/or interactive transcription systems are also described. They will exemplify the idea that the concept of input and output in a problem may depend on the subproblem we are interested or on the level of description we are talking about. These extensions and other examples motivate a section to discuss the limitations of the proposed classification and to point out the directions that, in our opinion, could be taken to overcome them.

figures of merit

It is also very important to analyze the behavior of the systems and how to measure their performance. A section is devoted to review some common assessment and performance measures associated to the problem types described before. Some figures of merit are very related to the concept of *loss* incurred when an action is chosen for a given, unknown, true state of nature. The concept of loss function will be considered in the next chapter when formalizing some machine learning concepts.

The following approach was proposed in the introductory chapter as a way to organize the existing know how accumulated in this area:

- to design a map of problem types;
- another map of solving techniques; and, finally,
- to relate them in order to know how to deal with a new problem.

showing limitations

This chapter is our humble attempt, for our narrow area of interest, to tackle the first map and also to show its limitations. Considering the repertoire of problems from previous chapter and the problem types recently described, a preliminary questionnaire to help us to situate a new problem or to characterize its relevant features is also sketched. It could be refined later when models and techniques are described. As expected, we will finish with some conclusions.

3.1 SOME PROBLEM TYPES

Our work is focused on statistical pattern recognition, where problems are formalized by means of statistical models which describe the relationship between a set of random variables. These variables take on values with certain probabilities and might or might not have the ability to influence each other (which can be expressed by means of graphical models, described in Chapter 4). Note that multi-modal systems, where different input random variables are associated to observations from different input modalities (captured with microphones, online pen devices, cameras,...) have nothing special from this point of view. The input modality is abstracted, meaning that they are rep-

multi-modality

1 Similar classifications can be found in other works (e.g. [Graves 2008; Section 2.3.1], although the repertoire of problems types does not seem so detailed).

resented in the same way and it is only the type of variables and the dependency between them what matters.

Machine learning is interested on how these models can be estimated from a finite set of data but, for the moment, we are just trying to describe some types of problems. A first approach to characterize most problems could be in terms of their respective inputs and outputs. To this end, let us see which are the most common types of variable values or attributes that these inputs and outputs may belong to:

types of random variables

NUMERICAL values which have an order relation and a distance. They are usually a subset of the real axis $\mathbb{R}$;

SYMBOLIC OR CATEGORICAL they are finite, discrete and have neither a notion or order nor distance.

There are other types of attributes such as periodic (e.g. the day of the week, which have a concept of distance but not an order relation) and ordinal (categorical values with an order relation but not a concept of distance). Also, values can be scalar or vector valued, can be grouped into (maybe heterogeneous) tuples, or even into more complex structures such as sequences, trees or graphs. These more complex types of values are known, in general, as "structured data". The following sections are related to types of problems associated with finite length sequences,[2] whereas more complex types of structured data (e.g. trees, graphs) are limited to the role of descriptions associated to sequences[3] or as part of the process to obtain sequence descriptions (see, for instance, Figure 53).

structured data

The approach consisting in characterizing problems in terms of their respective inputs and outputs is not the only possible approach and it is not exempt from drawbacks. For example, the same statistical model can be used to describe different problems of the proposed classification depending on which subset of variables is observed and which subset has to be estimated to obtain an output (or, generally, to take a decision). Reasoning about the probabilities of one subset of random variables given values of another subset[4] is known as "probabilistic inference" and will be addressed in the following chapters. Now, let us review some well known problem types related to sequences.

probabilistic inference

3.1.1 Sequence classification

The input is a variable length sequence, but it is assigned a unique label. Some examples of this problem type are isolated word speech recognition and music identification (when a fixed set of styles is considered, e.g. [Weinstein 2009]). As shown in Chapter 2, input patterns are usually represented as a sequence of fixed dimension feature vectors or frames.

2 We prefer to address finite length sequences to avoid the case of continuous streams. Nevertheless, many techniques and problems for streams are quite similar.

3 For example, trees are used as structural descriptions of many parsing formalisms as described in Chapter 5.

4 Which is not necessarily the complementary, since there may remain variables we are simply not interested in.

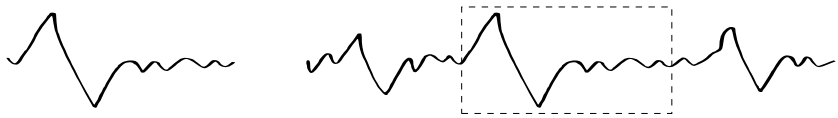


Figure 23: Difference between sequence and segment classification: The sequence of the left is identical to the segment of the right but, in the last case, the context can also be taken into account.

Although we are not focused on solving techniques for the moment, let us remark that there exist many different techniques to tackle sequence classification. It is possible, for instance, to compute a dissimilarity between sequences (for instance, by means of dynamic time warping), which allows the use of nearest neighbor classifiers. Note that both generative and discriminative methods can be envisaged in the statistical pattern recognition approach. An interesting feature of generative probabilistic models lies in the possibility of using sequence classification as part of more complex problem types in a sound way.

It is also possible to convert sequence classification into the simpler "fixed dimension classification" by transforming the input into a fixed length representation, which is described in Section 2.9.2 relating how to represent a variable length sequence in a fixed dimensional vector. This approach, known as *holistic*, has been applied to handwriting word recognition where words are considered as single, indivisible entities instead of dividing them into characters [Madhvanath and Govindaraju 2001; Castro *et al.* 2005; Ruiz-Pinales *et al.* 2007].

holistic word recognition

We can observe a limitation of the description of a problem taking into account uniquely inputs and outputs: they do not characterize the problems by how they should be solved, since some problems which may fit in the "classification of sequences" type should rather be tackled with methods more complex than others. For instance, determining only the language which has been used in a particular handwritten text line should probably require to recognize the handwritten words despite the fact that the final output is just a label (a language) from a finite repertoire.

3.1.2 Segment Classification

This problem is very close to the previous one. The sole difference is that, in this case, the sequence to be classified is a segment of a possibly larger sequence as described in Figure 23.[5] This brings the possibility of using a context which may comprise the entire sequence or, perhaps, just the surrounding parts of the segment to be classified. The importance of context to classify a segment is illustrated in Figure 2.

segments have context

There is an even more general problem which consists in providing an even richer context information. For instance, to classify a segment knowing not only their boundaries but also the labels (and, maybe, the boundaries) of surrounding segments. These extensions are addressed, as part of a more complex problem, in Section 4.2.1.

5 Is can be considered as a generalization of the previous problem since an entire sequence is also a segment of itself.

3.1.3 Joint classification

By "joint classification" we mean the classification of several items which should not be classified independently. Other works use the expression "structured label" when some inter-dependency structure between different parts of the output can be exploited, which we contrast[6] with the simpler case of "sequence classification" when a sole label, lacking internal structure, was obtained. Joint classification problems can be further divided into the following ones:

Fixed Dimension Joint classification

Involves classifying a fixed number of items which are, in turn, represented by a fixed dimension feature vector each one. This problem type can be reduced to the case of a fixed dimension classification by considering the output as a sole label taken from the Cartesian product of their corresponding output spaces although, in that case, the number of training samples needed to estimate a model might grow very fast due to the curse of dimensionality. Conversely, some problems which are usually considered a simple classification from a fixed dimension feature vector might be factorized into several independent labels.

Variable length joint classification (sequence labeling)

A more complex problem appears when the number of things to be classified is not known before observing the pattern. Instead of defining a model for each possible pattern size, it is better to describe a general procedure.

sequence labeling

"Sequence labeling" is defined, in some works, as the task of assigning a label to each element from an input sequence, which perfectly fits the current problem type. Unfortunately, the term "sequence labeling" can also be found referring to other similar, albeit not identical, problem types. Therefore, some caution is required when using this term in a too precise way.

An example of sequence labeling is POS tagging, described in previous chapter, where every lexical item (usually a word) is assigned a POS tag in a 1-1 correspondence. Clearly, the POS tag assigned to a word may depend on the tags associated to other words. It is also obvious that the number of words may differ between different sentences. Observe that POS tagging may be approximated by the previous "segment classification" problem if we assume that the neighboring words provides context enough to classify the current word no matter the POS tags of the context words. The fact that each word could be independently classified, under this approximate assumption, does not mean at all that labels were independent.

6 Unfortunately, some authors oppose "structured learning" to "i.i.d. learning", which seems misleading to us: we deal with sets of sequences obtained in an independent and identically distributed (i.i.d) way, but note that an i.i.d set of sequences is not the same as sequences of i.i.d. labels, hence we will avoid the use of "i.i.d. learning" to avoid confusions.

Joint classification of a segmented sequence

Once a signal has been unambiguously segmented, the problem of classifying each segment is a particular case of the previous problem type, excepting the fact that each individual classifier is a variable length classifier (or a segment classifier if we take the segment context into account). This case can be considered as an extension of the previous problem type. An example is described in [Sung *et al.* 2007]:

> *Phone classification is one of the simplest tasks in speech recognition, in which we are given a pre-segmented sequence of observations which must be assigned a single phone label.*

3.1.4 Segmentation and over-segmentation

Segmentation basically consists in describing the input sequence as a set of non-overlapping and non-recursive segments where the label of the segments is *irrelevant*.[7] An example of this problem is "word division" described in Section 2.1.

There exists some variants and extensions to the segmentation problem. Some additional restrictions can be included such as, for example, to fix the number of segments.[8] An interesting extension described in this section consists in providing several segmentations, which is known as over-segmentation. In some cases, each segmentation of this set can also be assigned a probability or a weight to denote the fact that some segmentations are more likely to be the correct one than others. In other contexts, over-segmentation cannot weighted and is primarily used as way to prune the search.

over-segmentation

Over-segmentation is a useful way to represent segmentation ambiguities to reduce the effect of the "pipeline problem" described in Section 2.9.3. For instance, some methods to deal with the joint segmentation and classification task perform a segmentation stage followed by a recognition of each segment. In that case, any segmentation mistake is usually irrecoverable.

Although our current classification is mainly based on describing problem types in terms of inputs and outputs, segmentation techniques can be classified into:[9]

---

7 Otherwise, the problem is known as joint segmentation *and classification*, to be described later. Unfortunately, we can find nomenclature problems everywhere, and segmentation is not an exception: in some works, such as in [Sarawagi 2006], segmentation is not only defined so that each segment is also assigned a label but also *"Conceptually, a segment means that the same tag is given to all elements"*, which poses some conceptual problems on consecutive segments which are assigned the same label.

8 Also related to the problem of alignment described later.

9 Unfortunately, this is another example of the lack of uniform nomenclature. For example, [Dunn *et al.* 2000] describes two approaches also called external and internal segmentation which are not related to our classification which is, needless to say, widely known and proposed in other works.
Another lack of uniform nomenclature example is [Bengio *et al.* 1995] where the term "INSEG" (INput SEGmentation) is used to refer to "external segmentation" and the term "OUTSEG" (OUTput SEGmentation) to refer to "internal segmentation". The same authors use the term "HOS" (Heuristic OverSegmentation) in place of "INSEG". All these expressions are not widely employed but can be found, nevertheless, in some works.

EXTERNAL when an explicit, usually task dependent, technique is applied. External segmenters are also known as "local" when the decision about the presence of a segment boundary is based on information surrounding the frame boundary. For instance, an external segmenter for extracting speech segments may proceed by measuring spectral differences between neighboring frames at both sides of each frame boundary. This would naturally lead to a local external over-segmenter. On the other side, even if the same spectral difference measures are used but we proceed by agglomerative clustering of frames to construct segments, the resulting method could no longer be qualified as local since the entire sequence should be a priori available. An advantage of local external segmenters is their ability to work "on the fly", which allows their use in online systems where information can be processed while it is produced;

INTERNAL denotes the fact that the segmentation is obtained as a *by-product* of a joint segmentation and classification task. For instance, in order to segment an spoken utterance into words, the most probable transcription and its associated segmentation are usually computed at the same time. Hence, it suffices to discard the transcription. Some segmentation and classification systems use an internal segmenter based on *another* simpler model which is applied before, which is quite similar to multi-pass systems. Indeed, the segmenter is a different module that can be seen as a black box and the fact that this segmenter is internal or external is abstracted.

Segmentation can be reduced to the previous problem type "Variable length joint classification (sequence labeling)" by labeling the frontiers between each adjacent or consecutive items as being or not a boundary between two consecutive segments, as was the case of BIO tagging described in Section 2.3. This reduction is usually associated to external over-segmentation techniques with a particular way to specify the set of segmentations, namely: by specifying, at each frame boundary, the possibility of a segment boundary. The set of segmentations is implicitly defined, in this way, as the segments compatible with those segment boundaries, which has the form of a "complete DAG" (there is an arc from each node to all the following ones) given the segment boundaries as nodes. In this work, we will extend this idea by labeling segment boundaries with additional information relating the restrictions on the set of segmentations. For example, we can determine, for some HTR tasks, that a sufficiently wide space corresponds to a separation between words. The frontier information produced by the over-segmenters described in this work can be classified into the following types:

extensions to over-segmentation

- a non-sure frontier, which means that the over-segmentation may contain solutions with segments that cross this frame frontier (segments that start before and finish later) and, at the same time, there can be segments also to be the begin or that end at this point;

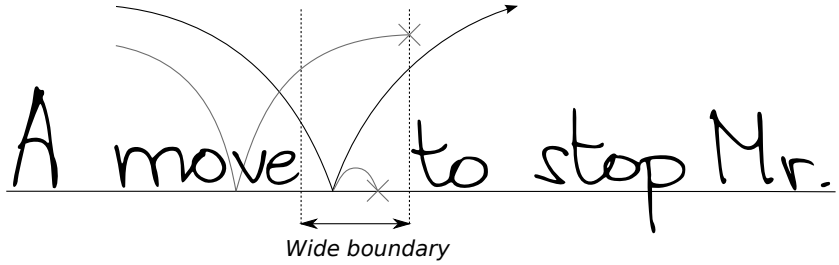


Figure 24: Example of a *fat sure frontier* over-segmentation label associated to a group of contiguous frame frontiers. Some decoders and graph generation modules proposed in this work can tackle them.

- a sure no-frontier, which means that no segment is allowed to finish or to start at this frame boundary;
- a sure frontier it a point at which all possible segmentations must have a segment boundary. This implies that there cannot be segments crossing this frame boundary;
- a *fat* sure frontier is an extension described into more detail in Chapter 11 and illustrated in Figure 24 where we can assure that there is a sure frontier in a contiguous group of frame frontiers without specifying in which one is the frontier. This means that segments starting before the fat sure frontier and finishing before are not allowed.

Note that these frontiers can be conditioned to some particular labels of the segments adjacent to the corresponding frontier.

3.1.5 Recognition

The problems of recognition and parsing will be addressed into more detail in Chapter 5, but we would like to include them into this enumeration for the sake of completeness. Both of them can be related to the concept of formal languages, which are sets of sequences from a given alphabet.

The recognition problem consists in determining whether or not the input sequence belongs to a given formal language. Recognizers or acceptors produce a binary output. This term is also used for decoders associated with the joint segmentation and classification problem described described in Section 3.1.15. In this case, they usually provide the sequence of hidden or latent variables or labels, as described into more detail in Chapter 4.

3.1.6 Parsing

The concept of parsing not only relies on a formal language as a set of sentences but is related to grammars viewed as finite ways of describing possibly infinite languages by means of some auxiliary structural descriptions. Parsing retrieves the set of those structures compat-

ible with the input sequence, which differentiates them from recognizers, although some recognizers can be converted into parsers.

Perhaps, the most known classical parsing algorithms are related to context free grammars (CFGs). These grammars are described in Section 5.4, and their description is usually based on derivation trees. Nevertheless, it is possible to describe these grammars and their corresponding parsing algorithms without resorting to those trees. In this way, Chapter 5 emphasizes the use of systems of equations over weighted languages to describe weighted languages defined by grammar formalisms and the use of semiring weights to describe the result of parsing (some semiring types can describe sets of parse trees). We refer the reader to Chapter 5.

3.1.7 Language modeling

This problem consists in estimating the a-priori probability of a sequence, usually a natural language sentence. This problem type is related to recognition and parsing but here we deal with stochastic languages, a particular type of weighted formal language.

Although this problem type is important by itself, it can be used as part or as a subroutine of others, as is the case of the joint segmentation and classification of sequences (Section 3.1.15). Their use in this problem type is motivated by the application of Bayes theorem in order to decompose the decoding process of the two stage generative model described in Section 4.3.

Most known language modeling techniques use the chain rule decomposition to reduce the probability of observing a sequence into the estimation of the probability of emitting a given word given the previous prefix, which could also be considered a particular type of prediction.

Given the importance of language modeling in this work, two entire chapters (Chapters 6 and 9) will be devoted to it.

3.1.8 Prediction/Forecasting

The aim of prediction consist in estimating the probability of possible continuations of a sequence. That is, the input is a prefix sequence and the output is either the possible continuation (hard decision) or the probability of each possible continuation. As stated in previous section, most language models are converted into this one by means of the chain rule decomposition. The term "prediction" or forecasting can also be applied to time series in general, not limited natural language processing tasks.

3.1.9 Translation

Translation receives a text or sequence from a source language and produces an equivalent sequence (conveying same meaning) in the target language. A very brief overview of Statistical Machine Translation (SMT) systems was described in Section 2.4.

3.1.10 Alignment

Alignment is defined as follows:[10]

> *to bring (components or parts, such as the wheels of a car) into proper or desirable coordination or relation*

The concept of alignment appears in at least three related, albeit slightly different, problem types:

- alignment of sentences and words in parallel texts (also known as bitexts) is used in many statistical machine translation (SMTs) approaches;
- transcription alignment consists in determining the segmentation of a signal provided their transcription;
- sequence alignment consists in identifying regions of similarity between several sequences. In this case, both signals belong to the same domain and a measure of similarity between regions is used. This problem type appears in some bioinformatic tasks.

The term "alignment" may be inappropriate in some cases and some of the problem types related to it are also described using other nomenclature: On the one side, the term alignment usually implies that the aligned elements appear in the same order in both parts (monotone alignment), but this is not necessary the case in bitext alignment. On the other side, the transcription alignment is also known as segmentation. In this way, the term "segmentation" seems overloaded and does not allow us to distinguish between the cases where the number of segments or their respective classes are unknown or not. Let us see the three sub-problems into a little more detail.

Parallel Text Alignment

SMT systems are based on the use of a large set of previously translated text also known as parallel texts or bitexts. The purpose of parallel text alignment is to establish links between the corresponding parts of each side of the parallel text. This correspondence can be done at different levels of granularity, as depicted in Figure 25 and, indeed, the alignment may proceed at different levels of granularity starting from the coarser elements (e.g. first paragraphs, followed by sentences and finished by phrases and words). The alignment is usually applied to large corpora so that unsupervised approaches make sense.

The general alignment description between two sequences should consider two cases, depending on the possibility of aligning groups of elements of each sentence or just individual elements. When we are limited to align individual components (e.g. the case of word alignment) it suffices to link or match positions and the result of this process is a bipartite graph which can be described as a matrix of a set of links as depicted in Figure 26. On the other side, when a group of several elements of one sequence can be related to another group in the other sequence (e.g. the case of phrase alignment), we should

10 See `http://www.wordreference.com/definition/alignment`.

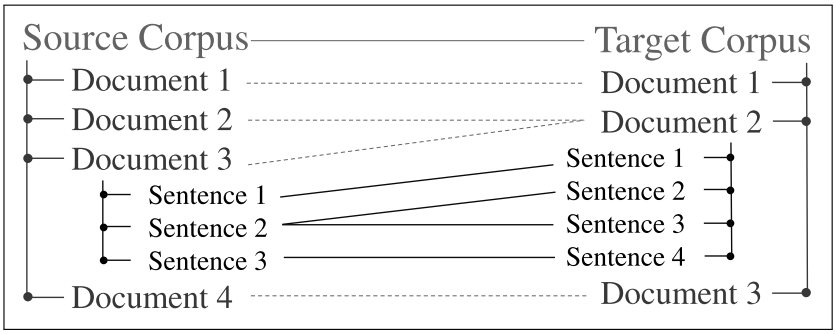


Figure 25: Hierarchical structure of a bitext and alignments at each level in the hierarchy (Figure taken from [Tomeh 2012; Fig. 1.1]).

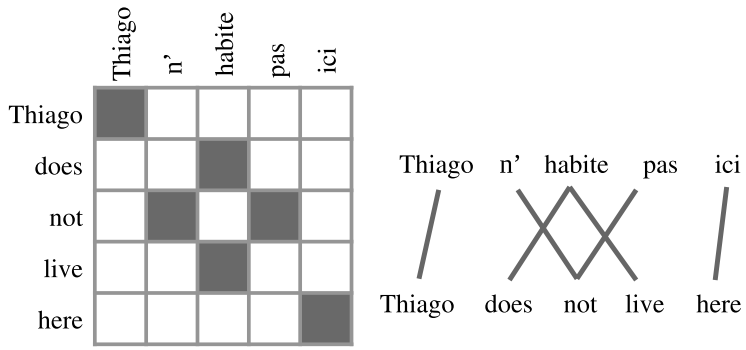


Figure 26: Example of word alignment. Two equivalent representations of a word alignment: matrix (left) and links (right) (from [Tomeh 2012; Fig. 1.7]).

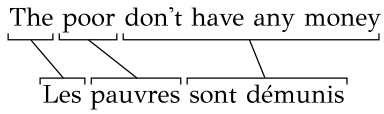


Figure 27: General alignment (inspired by from [Brown *et al.* 1993; Fig. 3]).

first group elements of each side. Note that this group, although usually denoted by the term "segment", might not be continuous (i.e. to include gaps). An example of such an alignment is illustrated in Figure 27 where several words of one sentence are connected to several words from the other sentence.

The result of the alignment process is not restricted to be just an alignment: several alignments accompanied with a weight or score (e.g. a probability) are also common. Since the set of possible alignments is so huge, a set of constraints is usually imposed to constraint this search space. It is possible to consider only one-to-many or many-to-one alignments (asymmetric alignments), or to assume that paired segments from source and target sequences appear in the same order (monotonicity). For a good introduction to this problem type, we refer the reader to [Tomeh 2012; Chapter 1].

Transcription Alignment

In transcription alignment problems we have the observed signal and the underlying label sequence at our disposal, but not the segmentation points (the alignment between both sequences). In the case of speech, the recording and what has been said is available but not the temporal marks indicating the precise position of words or phones in the recording. This information can be useful in several tasks:

- to synchronize subtitles;
- as part of the training of acoustic models in ASR;
- phonetic unit selection in text to speech (TTS) systems.

Transcription alignment seems a particular of the general case described before since it is restricted to many-to-one monotone alignments.

We will see that the transcription alignment, using perfect transcripts, is a much simpler tasks than the joint segmentation and classification problem described below. The language model is not required and, in some cases, the use of segment models are not required either. An example is the case of aligning handwritten text lines into words where it is possible, in some cases, to bypass character models and rely just on gap metric gap metrics. The handwritten text line is modeled in [España *et al.* 2004] as a sequence of ink and blank items obtained with a vertical projection of the line in order to apply a dynamic programming alignment algorithm, the same ideas are described more recently in [Zinger *et al.* 2009]. In other cases, the alignment is obtained together with the estimation of models describing the different segment types: transcription alignment is part of the process of training segment models (see Chapter 4) and, in this case, several pairs of sequences are usually required.

Sequence Alignment

Sequence alignment [Rosenberg 2009] consists in identifying regions of similarity between several sequences. In this case, a measure of similarity between regions is used. This problem type appears in many bioinformatics tasks, such as the study of the homology.[11] Some definitions of similarity allow the efficient implementation of sequence alignment based on dynamic programming but it should be remarked that the fact that a solution is computationally efficient does not imply that it is biologically plausible.

pairwise vs multiple

Sequence alignment is usually defined between pairs of sequences, but it is also interesting in many cases the alignment of multiple sequences.

global vs local

When we try to align the entire sequences, the process is known as global alignment. An alternative consists in trying to align subsections of the sequences. This makes it possible to align some parts while others remain unaligned. Local alignment has some applications in database searching.

11 Homology, as defined in `http://en.wikipedia.org/wiki/Homology_%28biology%29`, is the existence of shared ancestry between a pair of structures, or genes, in different species. Homology explains similarities which are due to inheritance from a common ancestor and not due to convergent evolution.

3.1.11 Keyword spotting, indexing, querying by example

Keyword Spotting (KWS) consists in detecting the presence of a segment, in the input sequence, associated to a particular pattern. This problem type has several applications such as information retrieval, the classification of mails containing or not the word "urgent", etc. Several issues can be taken into account:

- are we only interested in determining whether or not the keyword appears or do we require to know the position of the pattern inside the sequence?
- is it required to determine the number of times the keyword appears in the sequence or just its presence?
- the output of keyword spotting is a hard decision or some type of confidence measure? The main advantage of a soft decision is that we can later use it to make decisions taking into account precision/recall trade-offs;
- are we looking for just one keyword or for several ones? is it sufficient to determine if at least one such keyword appears or are we interested in determining which one? are we looking for the set of sequences (or documents) where the keyword appears? (e.g. performing the search over several sequences)
- are we probably interested in several KWS searches over the same sequences/documents? are sequences known beforehand?
- is the pattern or keyword to be detected given as a label sequence or by means of an example or template?

The last question allows us to distinguish between:

QUERY-BY-STRING when the keyword is given as a label sequence (e.g. orthographic or phonetic transcription);

QUERY-BY-EXAMPLE (a.k.a. sub-sequence matching) the keyword is a template (e.g. a word image in the case of handwritten data, a speech utterance in the case of speech recordings).

The term query-by-example is much more spread since KWS means, by default, query-by-string. Unfortunately, the nomenclature for specifying problem types is not as standard as we would like[12] and some authors also distinguish between:

EXAMPLE-BASED some type of distance[13] is used between a template and the segments of the sequence where we are looking for;

LEARNING-BASED implies that some machine learning techniques are used to train models.

As can be observed, a query-by-string is more suitable for learning-based techniques but it is also possible to devise an example-based approach by artificially creating templates associated to the query string.

12 This situation is aggravated by the fact that we are dealing with problem types in a task independent way and speech, handwritten, bioinformatics, etc. seem to use slightly different nomenclature.

13 This is limited to tasks where both signals are comparable with some similarity criterion. It is not possible, for instance, to directly search for a handwritten word occurrence in a speech recording and it is not obvious that the same handwritten word, written in a modern style, could be fairly compared with an ancient document or even with the style of other modern writers.

	User need known at indexing time	User need known at search time
System finds mentions	keyword spotting	spoken term detection (STD)
System finds relevant content	classification filtering	spoken content retrieval (SCR)

Table 1: KWS related tasks broken down along two dimensions (adapted from [Larson and Jones 2012; Table 1.1]).

Conversely, it is possible to use learning-based approaches in a query-by-example task by first recognizing the example in order to convert it into a label sequence representation.

Besides keyword spotting and querying by example, there exists other KWS related terms such as spoken term detection (STD) or spoken content retrieval (SCR), which are clearly ad hoc names for some speech tasks. The classification of these tasks not only depends on what are we looking for or what are the inputs and the outputs but also when this information is known. In some cases, the keyword and the sequences/documents are known at the same time, but in other cases it is possible to apply a two-stage approach where we can distinguish between:

INDEXING TIME (or pre-processing phase) is the time elapsed between the sequence is known and before the term or the keyword to be searched is provided. We can invest some effort and resources[14] during the indexing time to speed up the future searches, this makes sense particularly when several searches are expected over the same sequences;

SEARCH TIME is when the keyword is available. We can take profit of preprocessing done during indexing time.

According to [Larson and Jones 2012; Section 1.4], we can classify KWS related terms into four classes depending upon how the system takes into account the user requirements and when the provided information is available, as illustrated in Table 1. A task related to STD is spoken utterance retrieval, where we are trying to find the set of documents where a keyword appears. The distinction between finding mentions and relevant content is very tasks dependent and seems mostly related to natural language processing tasks.[15] We will restrict our focus, in the remainder of this section, to the "finding mentions" case. Let us also remark that these terms are not exempt from nomenclature ambiguities and the difference between KWS and STD is not only limited to the difference between indexing and search time but also to the length of the keyword pattern: KWS for short terms and STD for terms possibly composed by several words. Otherwise stated,

14 The amount of work and the type of information generated during the indexing time may vary from just some pre-processing up to pre-computing all possible word occurrences.

15 An even more semantic task, also related to information retrieval, is "Question answering" where the system tries to answer to a question formulated in natural language.

there are techniques described as KWS but based on two-stages. Indeed, the expressions "online KWS" or "on-the-fly KWS" are usually employed to emphasize the fact that sequence and keyword are processed at the same time.

Keyword spotting is an example of how a problem can be converted into another one from a different type. A joint classification and segmentation of the sequence can be performed by using a lexicon and a language model. The probability of occurrence of a keyword in a sentence can be computed by marginalizing all output outcomes which contain these word and all these outputs can be efficiently approximated[16] by means of word graphs and forward-backwards dynamic programming algorithm (which can be customized for the marginalization process). It is also possible to include additional restrictions such as the presence of a keyword in a range of the input sequence. A drawback of this technique is that it requires more knowledge than just the keyword model in the form of lexicon and language modeling and may be less robust when faced with words out of vocabulary (OOV) and noises. These problems may guide the decoding process with a wrong hypothesis. Note that this technique, related to the time dependent posteriors of Figure 31, is much more appropriate to obtain a posterior probability of occurrence than a simple search of the word over the output of a recognition system. Let us also remark that applying keyword spotting with different keywords over the same sentence produce measures which are not independent among them, careful attention should be paid to this fact.

marginalization

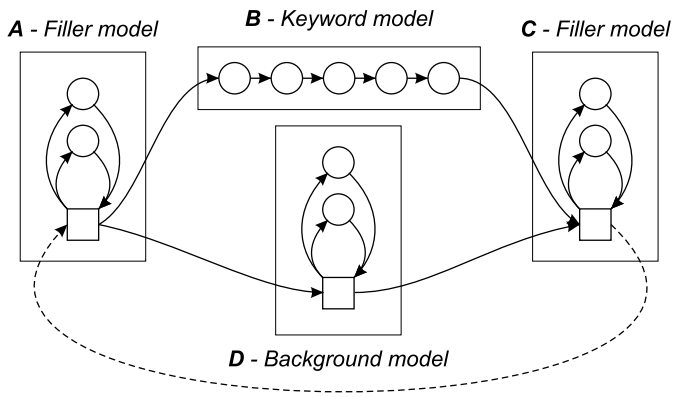


Figure 28: General scheme of KWS (taken from [Szöke *et al.* 2005; Fig. 1]).

A simpler and computationally cheaper technique requires only two different models: the keyword model and a background or garbage model used to model the non-keyword part of the sequence [Rose and Paul 1990; Fischer *et al.* 2012]. Although the non-keyword (background or filler) model is general enough to recognize the keyword, it is expected that the keyword model, constructed *ex profeso* for the given keyword, would produce a better score than the background model. In this way, both models compete as depicted in Figure 28 where we have to assume that B and C process the same segment of the sequence

filler approach

16 Because of pruning techniques.

so that the final likelihood[17] of the parts ABC and ADC only differ by the use of a different model on the same segment. The likelihood ratio is usually used as a confidence to detect keywords.[18]

lexicon-free

A great advantage of this technique is that it is lexicon free and language independent since both the lexicon and the language model are no longer required. There is a risk in this keyword spotting approach when it is possible that a segment associated to a pattern can be also correspond to the accidental concatenation of parts of other patterns. In the case of keyword spotting on speech, it may require to take into account the acoustics, the phonetic transcription of the lexicon and the language model. The keyword spotting by means of recognition may solve the problem of accidental concatenations since recognition addresses problems such as homograph disambiguation and takes the whole sentence into account, but this also requires more accurate models and is more costly. In spite of its simplicity, the filler approach requires to recognize with an ad hoc model for each different keyword. Some two-stage approaches create a word graph either of words or sub-word units during the indexing time.

accidental concatenations

meta-problem

A novel *meta*-problem (we have not found references in the literature) is the study of the probability of accidental concatenations in the case of naive word-spotting methods based on looking for the keyword ignoring the context. The technique would estimate the probability of the accidental combination of sub-word sequences leading to the keyword and would also consider the language model to estimate the probability of appearance of words leading to these sequences. This meta-problem would be useful to choose which technique is best suited in each case and would help to improve filler models to each particular keyword to reduce the a-priori probability of an accidental wrong detection.

A problem related to keyword spotting appears in spoken language understanding tasks where the speech understanding systems tries to fill some slots of a semantic frame by performing some type of "concept-spotting" where words not related to slots are just skipped. Also, there exists some proposals to use KWS to construct recognition systems [Fousek and Hermansky 2006].

3.1.12 Unsupervised term discovery

Unsupervised term discovery (UTD) is the task of discovering recurrent patterns in a set of observed sequences. This is an unsupervised task in the sense that no repertoire of hidden labels and no lexicon are provided. In the particular case of spoken term discovery, and although this task is related to the study of language acquisition, the concept of terms are not necessarily associated with any kind of linguistic counterpart such as words. Nevertheless, one of the interest in these unsupervised approaches is to address the problem of out-of-vocabulary (OOV) words in ASR.

17 These terms and models will be described in Chapter 4. It is not easy to serialize such a large state of the art without incurring in forward dependencies.

18 Another approach consists in applying Viterbi decoding and checking if the output sequence contains the keyword.

Unsupervised term discovery is more complex than the clustering of a set of sequences since, now, the positions of the terms are not provided. Although a brute force solution would compare every possible sub-sequence pair, more reasonable strategies would apply different kind of pruning and optimization techniques, reuse computations and so on. The information gathered by this pairwise comparison is then used by a clustering stage.

There exists problems similar to the discovery of repeated patterns such as the one consisting in finding the unusual sub-sequences, which is related with the detection of anomalies.

UTD can be used as a part or subroutine of other tasks, usually related but not limited to data mining. This task plays also a relevant role in bioinformatics. It is useful in ASR, for instance, not only to deal with OOV words but it is also useful in models of language acquisition, to find the best repertoire of sub-lexical units and, in general, to cope with languages with very limited resources.

As with spoken term detection and with query by example, UTD is based on similarity measures between sequences since, in all cases, repeated patterns must be similar but not necessarily identical. Besides a relevant similarity measure, other a priori information usually provided as part of the problem setting is information about admissible segment lengths in order to avoid the discovery of too short segments.

3.1.13 Filtering/Denoising and Smoothing

Given a sequence which we can assume that has been transmitted by a noisy channel, the aim of denoising consist in separating the original sequence (what is considered the meaningful signal) and the remaining noise. State-space models are a family of generative models which can model the production of the observed signal. In these models, the internal state cannot be directly observable but is used to infer the next state and the observed signal. Filtering, in this context, consists in estimating the state distribution at time t given all the observations up to (including) t.

The concept of smoothing is quite similar since it also tries to estimate the state distribution at time t although, in this case, the entire observed sequence is available.

A related problem is blind source separation where several signals are mixed together and the main aim consists in recovering one of the components as in the cocktail party problem where several persons talks simultaneously in the same environment.

3.1.14 Summarization

Summarization consists in extracting a shorter description of the topics addressed in the input source. This seems more a particular task than a problem type.[19] There exists a particular approach to summarization which includes an additional constraint: the result of summarization is basically made of sub-sequences from the original source.

19 In terms of inputs and outputs, it is not very different from translation.

3.1.15 Joint Segmentation and Classification

Sayre's paradox

This problem consists in jointly performing a segmentation of a signal and a classification of the resulting segments. It is related to the Sayre's paradox [Sayre 1973], described into detail in Chapter 1, which states that *classification requires segmentation, but segmentation also requires classification.* In that introduction, an idea to break this "chicken-and-egg" circular paradox was considered: to take all hypotheses about segmentation into account and to choose the segmentation maximizing an objective function. Observe that, when a segmentation is chosen, the problem seems like the "Joint classification of a segmented sequence", but there is a subtle difference since, now, hypotheses associated to different segmentations have to be compared. This difference explains why the use of conditional or discriminative methods can be probably applied easier to the joint classification of a segmented sequence than to the joint segmentation and classification. Nevertheless, since we are not dealing with the precise methods to solve those problems yet, we defer this discussion until Chapter 4.

The basic problem, unless otherwise stated, consider that the input is a sequence and the output is another sequence of labels. Nevertheless, there may exist other possibilities.[20] Let us see what can be considered as inputs and outputs in this problem type.

Inputs

Relating the inputs, the following extensions can be used independently or at the same time:

OVER-SEGMENTATION information can be used to extend the input. This additional information may restrict the set of allowed segmentations of the input sequence. This extension comes without loss of generality since it is possible to label each frame boundary with "non-sure-frontiers" leading to the original case. This extension makes the "joint classification of a segmented sequence" and the "joint segmentation and classification" problem types as opposite points of a continuous spectrum. Although over-segmentation is usually applied as a pruning technique to improve the performance of decoders, segmentation constraints may improve not only the performance but also the accuracy in some problems such as interactive transcription (to be described shortly) since it is possible for the user to provide this kind of information;

INPUT DAG Another interesting extension to this problem consist in providing not only a sole input sequence but several alternatives in order to represent preprocessing ambiguities, as described in Chapter 2. A compact way to describe a set of sequences consist in using a directed acyclic graph (DAG) or even non recursive context free models, which will be described in detail in Chapter 5. In this case, we are combining the contributions of all those

20 As described in Section 2.9.1 relating the underlying representation. In any case, both segment and landmark based approaches take a sequence (even a preprocessed frame sequence) as a starting point, so they should be rather considered related to the models or techniques used to solve a problem than to the problem itself.

sequences to the obtained sequence of labels or, when the best joint segmentation and classification is required, taking the best of all of them among the different input sequences implicitly defined.[21] The over-segmentation information can also be extended to the case of several inputs;

AUXILIARY FEATURES using the nomenclature from [Stephenson *et al.* 2004] which refers to those constant or slow changing features which are not associated to a given frame nor have a direct dependency on the segmentation of the observed sequence or the correlation with the hidden one. An example of these features is the rate of speech, the vocal tract length or the gender in ASR. It is possible, for some auxiliary features, to obtain estimates by some "external" (to decoding) means. In this way, this information can also be provided to the decoder.

Outputs

Relating the outputs, several levels of detail may be required:

1-BEST LABELS is the sequence of labels which more probably "explains" the observed sequence, which is usually known as transcription. Note that this procedure to obtain the best transcription is not the same as obtaining the best joint segmentation and classification and to discard the segmentation, although this last approach is usually assumed by most decoders;

1-BEST SEGMENTED AND LABELED corresponds to the best individual segmentation and classification hypothesis to explain the observed signal;

1-SEGMENTATION can be obtained when only the best segmentation is required,[22] we defined this procedure as "internal segmentation". As with the 1-best labels, we have two possibilities: either marginalizing the probability of each segmentation and taking the most probable one, or discarding the labels form the best individual hypothesis;

k-BEST LABELS sometimes, the 1-best hypothesis does not suffice, specially when this problem is part of another one (remember the *pipeline problem* described in Section 2.9.2). In that case, the k-best sequences of labels is a way to approximate the set of most probable transcriptions, since the accumulated probability mass of the k-best transcriptions reaches the unit quite fast in many real tasks. As in the case of 1-best transcription, we can accumulate the posterior probabilities associated to each possible transcription marginalizing over segmentations or, the most common case, this list is approximated by discarding the segmentations associated to the k-best most likely labeled segmented hypothesis.

21 Both alternatives roughly corresponding to the application of the same algorithm using different semiring types.

22 A common example is transcription alignment. Another less known example is to obtain a sequence of segments from a frame vector to be used in some segment-based ASR systems.

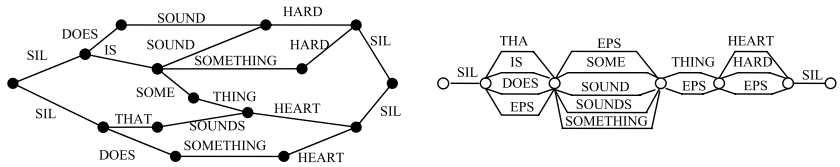


Figure 29: A word graph (on the left) and (right) the corresponding confusion network (figure taken from [Xue and Zhao 2005; Fig. 2]).

The number of solutions (value of k) is usually specified a priori, but some systems allow us to ask for more solutions on demand so that we can proceed until a condition is met, which is much more flexible. Also, sometimes the *set* of k-best solutions, without any particular order, may suffice. The computation of k-best lists is usually based on finding the shortest paths in a weighted graph, in this case, some algorithms [Eppstein 1998] provide specialized versions which are asymptotically faster when the ordering of the k-best paths is not required;

k-**BEST SEGMENTED LABELS** can be obtained when segmentation is also required. Note that in many tasks, specially in ASR, there can be an overwhelming number of hypotheses with identical sequences of labels, differing only slightly in their segmentation;

k-**BEST SEGMENTATIONS** seems a trivial extension of the internal segmentation procedure described below. Nevertheless, it is usually easier to obtain "the set of segment boundaries appearing in the set of k-best segmented labeled hypotheses". That is, what we obtain is just a subset of frame boundaries, hence obtaining some segments which did not belong to the k-best segmentations;

WORD GRAPHS are, compared with k-best lists, a much more compact way to represent a huge list of hypotheses. They are directed acyclic graphs (DAGs) as illustrated in Figure 29 (left). They are labeled with lexical items (usually words) and scores, and constitute a particular case of weighted finite state models, which will be covered in Chapters 5 and 6. Note that two different scores are usually stored independently: the language model score and the model associated to the generation of the observed segment given the hidden label (e.g. the acoustic model in the case of ASR). Word graphs, as with k-best lists, may contain segmentation information. In this case, time marks are associated to the vertices. When the segmentation is not required, word graphs can be much more compact and it is possible to obtain them directly (without first generating the segmented version). Some issues relating both approaches will be covered in following chapters, aspects such as how to obtain posteriors, for each word at each edge, from word graphs by summing the probabilities of all paths containing them. To this end, the probability mass not covered by the word graph is usually considered as negligible and, consequently, ignored. This making it possible to normalize the scaled probabilities;

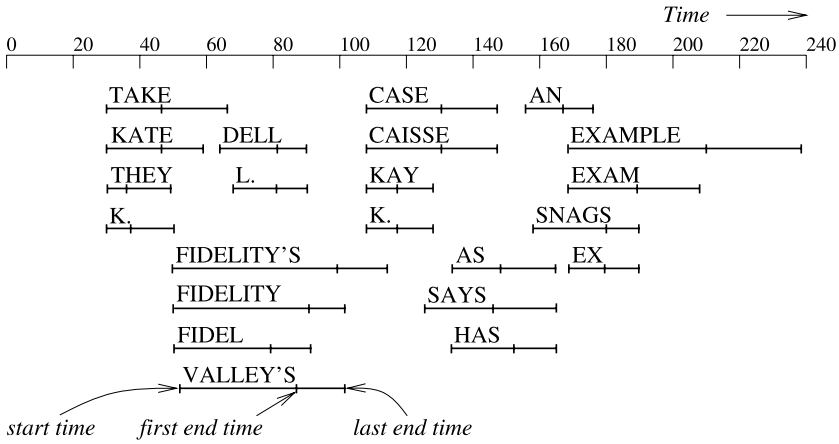


Figure 30: Example of a word lattice (figure taken from [Ravishankar 1996; Fig. 4.7]).

WORD LATTICES is a problematic and confusing term to describe another type of representation, since "lattice" is almost invariably used as a synonym of "word graph".[23] In this work we will avoid to use this term to refer to word graphs. The term "word lattice" as used, for instance, in [Tomita 1986; Chien *et al.* 1991] specifies a set of segments or spans with labels and scores associated to them, without any explicit graph structure, as depicted in Figure 30.

Word lattices allow gaps, as can be observed in Figure 30. This representation makes sense in some tasks where some type of segments are not explicitly represented, as is the case of silences and filled pauses in ASR, which would roughly correspond to the gaps.

The use of word lattices is not very extended compared with word graphs. Indeed, [Chien *et al.* 1991] describes how to obtain a word graph from a word lattice: Word segments are sorted in the order of their right boundary or ending points and two word segments u and v are considered connected if there is no other segment between them (that is, $u \leqslant v$ and there is no w such that $u \leqslant w \leqslant v$, when $x \leqslant y$ if $\text{end}(x) \leqslant \text{begin}(y)$).

Although they are not equivalent, we can find a resemblance of this representation and certain types of transcription as proposed in [Fischer 2012; Chapter 10], where the goal is described as follows: *"The goal of aligning inaccurate transcriptions is to automatically extract as many pairs of word images alongside with their correct transcription as possible"*.

A "word trellis" is essentially a word lattice. This term is used in [Lee *et al.* 1998; Kawahara *et al.* 2004] as the set of words which finished in each frame together with their scores and the corresponding starting time;

23 Most works use them as synonyms, others such as [Johnson and Harper 1999b] use them to distinguish between Mealy and Moore implementations of word graphs.

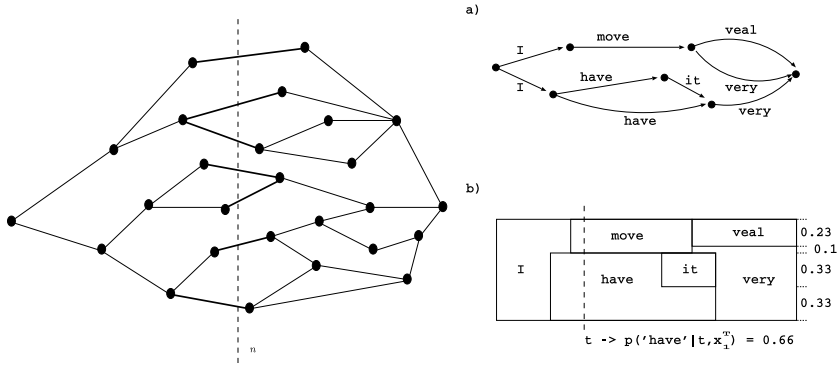


Figure 31: Time dependent posteriors (figures taken from [Evermann 1999; Fig. 3.6] (left) and [Hoffmeister *et al.* 2006; Fig. 1] (right) respectively).

CONFUSION NETWORKS [Mangu *et al.* 2000; Hakkani-Tur and Riccardi 2003; Xue and Zhao 2005] are a particular type of word graph with a particular topology which avoids overlaps and forces all competing words to be in the same group or "confusion set". Words in each confusion set are labeled with posterior probabilities. Moreover, confusion sets can also contain the empty word. Confusion networks are linear concatenations of these confusion sets, as depicted in Figure 29 (right), which explains why they are also known as *sausages*.

The transformation of a word graph into a confusion network is performed by a clustering procedure that groups overlapped segments into clusters based on their similarity[24] and their probabilities. The main motivation for using confusion networks it due to the fact that most decoders find the hypothesis that minimizes the sentence error rate[25] even if we are usually more interested in the word error rate.

Confusion networks can be used to produce a transcription which is known as "consensus hypothesis". This transcription is obtained by picking the most probable label from each confusion set. This transcription is expected to minimize the word error rate. Nowadays, confusion networks are used as intermediary representations in many other tasks.

Confusion networks are more compact than word graphs while trying to preserve as much as possible the recognition accuracy, although they are limited by the fact that all competing word hypothesis must be aligned, which poses some problems with words of different length.[26] Their particular topology allow the use of specialized decoding and combination algorithms [Evermann and Woodland 2000];

24 E.g. using Levenshtein distance between the respective word transcriptions. Levenshtein distance is introduced in Section 3.5.

25 Sentence error rate and word error rate will be shortly defined in Section 3.5.

26 The fact that confusion sets can contain the empty word may palliate this in some way. Some authors such as [Xue and Zhao 2005] consider the possibility of splitting long words.

WORD POSTERIORS PER FRAME is a representation which can be efficiently computed from the word graph by means of the forward-backward dynamic programming algorithm. This representation provides the posterior probability for each label (e.g. each word of the lexicon) at each point of the sequence, as illustrated in Figure 31.

Compared with word graphs, the boundaries of each word and the correlation among them are lost, but the posteriors per frame are interesting in many tasks since they provide the global confidence about the appearance of a word at a particular position.

This information summarizes what in the word graph would be different occurrences of the same word at slightly different boundaries. They can be used for decoding (as extra information or to improve the pruning), to compute confidence scores, to detect out of vocabulary words or to perform KWS.

Reducing other problems

Some of the previous problem types can be seen as restricted cases of the joint segmentation and classification task, let us briefly sketch how some of these problems can be solved with this type of problem, but note that this does not imply that this is the only or even the best way to solve them.

SEQUENCE CLASSIFICATION can be seen as a particular case of this problem, although would seem overkill, if we restrict the set of allowed label sequences to be of length one;

JOINT CLASSIFICATION of a segmented sequence is a particular case of the joint segmentation and classification when we restrict the possible segmentations to the case provided by the input;

(OVER)SEGMENTATION by means of *internal* segmentation techniques has been explained as a way to use the joint segmentation and classification to in this task;

TRANSCRIPTION ALIGNMENT, restricted to the particular case of a monotone many-to-one alignment to align the observed signal to the underlying transcription, can be performed by means of a joint segmentation and classification, since it suffices to restrict the allowed label sequences to the transcription provided as part of the input;

KEYWORD SPOTTING, as explained before, we be reduced to the joint segmentation and classification tasks in at least two different ways: by using an standard transcription system and computing the probability of appearance of the keyword in the best solutions, or by using a keyword model and a garbage model.

Some additional problems related to the joint segmentation and classification on sequences are described in Section 4.3.

3.2 SOME EXTENSIONS

Although some problem types described before required several input sequences (as is the case of parallel alignment), some problem types which usually receive a sole input sequence can be extended to include several ones. We have also shown that some additional information, in the form of auxiliary features, can be provided to the system. There are some interesting scenarios where these extensions make sense. For instance, when the output is to be post-processed by a user (or a set of users), it is possible to involve the user during the problem solving process leading to the interactive assisted paradigm, which may require to take into account the user feedback in the form of validated prefixes, repeated utterances, segmentation constraints and so on. This section reviews these extensions. A more general framework to deal with this variability will be addressed in Section 3.3.

3.2.1 Several input sequences (usually related to Multimodality)

Multi-modality basically consists in the capability to take into account information from multiple modalities (e.g. acquired by means of microphones, online pen devices, cameras, etc.). It arises in many practical problems, specially in some interactive systems (which will be reviewed in the following section). We saw, in the introduction of this chapter, that multi-modality does not require a special treatment when describing a model by means of random variables, since the input modality is abstracted. The fact that information received from multiple modalities is not independent can also be taken into account in the same way as with unimodal information.

Nevertheless, in the case of problems on sequences and their description in terms of inputs and outputs, we can observe that multimodal recognition is usually associated to problems with several input sequences. For instance, let us suppose that a reader play aloud a handwritten document so that the recognition system has both the handwritten lines and the associated speech recordings at its disposal. Although a little artificial, this is a clear example of a multi-modal problem related to sequences when it is known, in advance, that both input sequences have the same transcription.

does not imply multimodality

But note that these types of problems do not necessarily imply multimodality, as is easily observed with another example: a user calls an automatic dialogue system to book a room in a hotel. The system ask her/him for the name of the city and, if the confidence provided by the ASR system is low, the user is invited to repeat the answer. In this case, taking into account the information provided by both user utterances should be better than just using the last one [Cesari *et al.* 2008; Zweig 2009; Nair and Sreenivas 2010].

repeated utterances

But note that the extension of the joint segmentation and classification problem to the case where several input sequences have the same transcription is not as easy as could be expected: in the previous room booking application, the dialogue system should also consider the possibility that the user was trying to change the topic during the first utterance, so that the second utterance was not the repeti-

tion of the name of a city. Another slightly different scenario occurs when several persons are describing the same thing, paraphrasing a text or are freely interpreting a script. Although related, the transcriptions are no longer equal. How this correlation could be taken into account? The following section, devoted to interactive transcription systems, will show other examples involving several sources of information, some of them in the form of sequences and possibly, although not necessarily, coming from different input modalities.

The previous examples can probably be solved by means of independent ASR or HTR systems provided that they are able to compute the entire set of hypothesis (or an approximation in the form of k-best lists or word graphs), together with their respective scores, in order to combine them. In that case, the results from each recognizer can be combined using an approach known as *late fusion*. The combination of several transcriptions is similar to some ensemble methods approaches [Fiscus 1997] where different, hopefully not too correlated, recognition systems are applied to the same input and the k-best output hypotheses generated by different systems are combined to yield an improved overall recognition.

late fusion

The late fusion approach is reasonable for sequence classification tasks with a reduced number of classes, which makes it possible to obtain the entire output distributions, but it is not efficient for joint segmentation and classification tasks with a large vocabulary where even large word graphs would probably cover a tiny part of the space of output hypotheses. Joint decoding of several sequences may be a more reasonable approach, which can also take into account the (maybe only partial) alignment of different sequences. For instance, in the case of online HTR, we can obtain the offline counterpart in order to recognize it. This is not a good idea *per se*, since online data usually yields better results, but the combination of the online and offline recognition results would probably improve individual systems when they are not completely correlated. In that case, late fusion is not the best option since we can not only exploit the transcription but also the segmentation. Note that the online version contains stroke trajectories whereas the offline counterpart is broken into image columns, meaning that they do not have the same number of frames and that they are *not aligned at the frame level*. Nevertheless, decoding could be synchronized at the word level. Additional gains can be obtained, not only in terms of efficiency, when it can be assumed that any stroke is never shared by different words,[27] since it allows us to restrict the search by synchronizing input modalities in some particular points based on strokes (online modality) and connected components (offline counterpart). This type of restrictions relating over-segmentation will be addressed in the second part of this work.

partial alignments

In general, joint decoding is more efficient than late fusion because it makes possible to combine several data modalities during the recognition instead of being forced to wait until the end. More informed hypotheses allow better pruning techniques. Another similar example is the case of multi-stream, multi-tape or sub-band based ASR [Bourlard and Dupont 1997] where several features are obtained from the speech

multi-stream & multi-tape

27 This property, not generally applicable to HTR, holds for most handwritten styles.

signal and are used by different acoustic models which are not combined at the frame level but at the phonetic level. This is not equivalent to the approach consisting in concatenating all the features to obtain a larger frame to be used by a sole acoustic model. It is not equivalent to combine acoustic models at the frame level (as in [Barrault *et al.* 2008]) either. These are also interesting alternative approaches, but they remain less general since they are restricted to frame-wise aligned input sequences.[28]

3.2.2 Interactive systems

Fully automatic recognition systems almost never achieve the recognition rate performances required in real world applications. In those cases, human intervention seems mandatory to post-process and validate the output. The use of confidence measures and the possibility of the *reject option* may help in some cases by restricting the presence of possible errors the users might look for. It is also possible so skip the correction of some errors if a given error rate can be assumed thereby offering a trade-off between expected quality and supervision effort.

Another approach consists in taking the user feedback into account in the recognition process itself. The main aim of this integration is to reduce the user effort required to perform a given pattern recognition task[29] w.r.t. performing these tasks entirely by hand or compared to post-processing the output of an automatic non-interactive system. This human effort does not only depend on the time invested in the process but also on the physical and the cognitive effort. Indeed, the entire idea of a user effort which have to be reduced can be bypassed in at least two different ways: it is possible to create games that people will voluntarily play and which require some labeling or annotation task [von Ahn and Dabbish 2004]. Also, we can take profit of things users *have to do* in any case, such as the use of *captchas*[30] to transcribe handwritten words with a low confidence measure from a previous HTR [von Ahn *et al.* 2008].

labeling games

Let us also remark that, since the user effort is compared with post-processing or performing the task entirely by hand, it should be mentioned that this "baseline" may significantly differ depending on the use of proper annotation tools: there exists interactive annotation tools, not necessarily based on pattern recognition techniques, intended to reduce the user effort. If these tools may, no doubt, benefit from pattern recognition techniques, it is also true that any interactive transcription

28 The possibility of using multiple input streams which are correlated at some level, not necessarily at the frame level, is discussed in following chapters and is related to multi-tape models.

29 Usually transcription, but this approach can also be applied to word spotting or indexing, alignment, translation, semantic annotation in SLU tasks, and many other problem types including the generation of text as is the case of predictive text entry methods described in Section 2.6.

30 "captcha" is an acronym of "Completely Automated Public Turing test to tell Computers and Humans Apart". They are tests, usually based on HTR or ASR, that a person would solve without too effort but which would be too difficult to perform by a computer. They are commonly used to prevent unwanted automated software from performing some actions reserved to human users, such as registering to some websites or posting in some blogs. For more information, see `http://en.wikipedia.org/wiki/CAPTCHA`

system have to be useful even without pattern recognition capabilities: some annotation is usually required when new data is first available in order to train some models by means of supervised pattern recognition techniques. This would ease the production of more data to train improved models roughly following the pool-based active learning cycle [Settles 2010] or the bootstrapping approach. Nowadays, the use of semi-supervised and unsupervised methods aim at further or completely reduce this initial effort.

transcription bias

Relating the validity of manual annotation tools, we have to mention that some authors warns us that automatic transcriptions may influence the user transcriber hence producing a bias that should be taken into account [Bazillon 2011].

Interaction protocols

user model

Every interactive system has to define a set of allowed user actions and the corresponding set of expected system responses, what is usually known as a *user model*. Interactive assisted transcription systems are not an exception.

One of the most common approaches to interactively validate and correct the transcription of handwritten lines consists in proceeding only in a left-to-right order [Toselli *et al.* 2010; Vilar *et al.* 2010].[31]

We can assume, in this setting, that the user corrects the first encountered error so that, when a label (usually a word in a transcription) is corrected, all previous words are validated as correct and the corrected label is known to be incorrect.[32] With this new information, the system can re-estimate the words after the validated prefix. Since this new recognition process is better informed, the new suffix is expected to contain less errors.

Hopefully, some errors the user would have to correct in a more traditional post-edition approach have now disappeared, hence reducing the expected overall user effort. The optimal decision rule associated to this protocol is proposed in [Alabau *et al.* 2012]. Alternatives to the left-to-right user model, specially when the user effort is limited, include the presentation of word hypothesis depending on the confidence[33] in order to propose the supervision of words which are more likely incorrect [Serrano *et al.* 2013].

Input methods

As illustrated in Figure 32, different types of input methods can be used to give feedback to the interactive assisted transcription system, among them:

- keyboard;
- mouse or electronic pen to directly make gestures or to write directly over the image of the text to be transcribed (in the case of

31 [Toselli *et al.* 2011] uses the term "interaction protocol" to denote a more abstract concept than user model since a given protocol might be implemented by means of different user models. Left-to-right interaction protocol would be considered an interaction protocol (among others).

32 Who would correct an already correct word?

33 Related with the uncertainty sampling approach from active learning [Settles 2010].

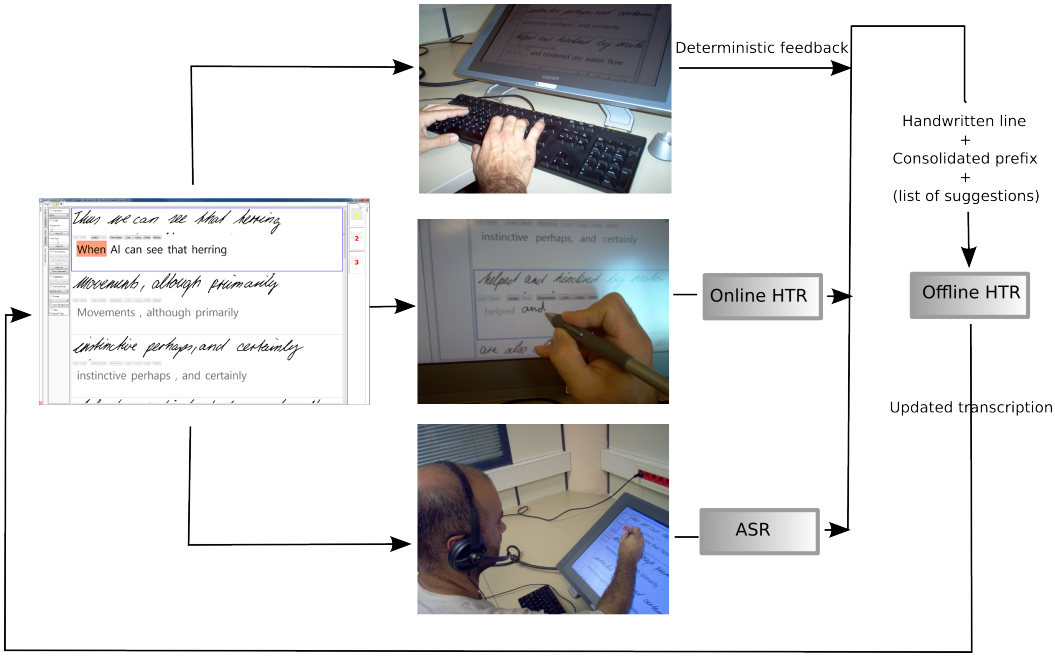


Figure 32: Screenshots of the multi-modal interactive assisted transcription tool developed at State and Hitita projects (the author made part of them). The user can correct a given transcription by means of a keyboard, a microphone (pronouncing the corrected word), a mouse or directly with an stylus (either to perform gestures or to write on the screen). Figure taken from [Castro *et al.* 2011; Fig. 1].

transcribing printed or handwritten text) or over the transcription;

- electronic pen can be replaced by fingers in some touch screens, and multi-touch screens allow sophisticated gestures with several fingers;
- a microphone in some scenarios can be very useful to give orders or to introduce the transcriptions or some words [Castro *et al.* 2011];
- the use of lip reading (webcam) can also greatly improve the efficiency of ASR systems;
- eye-gaze tracking devices, although not included in the figure nor in the current prototype, is also an input method worth considering.

With them, the user is able to:

- make gestures with an electronic pen or with the fingers to (the list is not exhaustive):
  - correct the segmentation of the line into words (either splitting several words incorrectly considered as being a sole word, or joining a word broken into several ones);

  - mark a word as incorrect (which provides less information than also specifying which is the correct label);
  - delete a word from the transcription;
  - delete some characters from a word in the transcription;
  - move the focus over a word in order to modify it by keyboard, speech (microphone), etc.

- the keyboard can be used not only to modify the transcription, but also to change the position of the cursor and even, as with the use of pointing devices or even the fingers, to select a word from a list (usually of k-best) of candidates;

- the electronic pen can also be used to replace some part of the transcription either by writing entire words or even just by modifying some graphemes of a word;

- the microphone can be used to introduce orders or to replace the keyboard or the electronic pen to modify part of the transcription. In this case, homograph disambiguation is required.

As can be observed, there exists many different alternatives. Some input data modalities, such as online handwriting or ASR, may also require to perform a recognition or classification process, whereas others, as is the case of keyboards, can be considered non ambiguous, error free (excepting user mistakes) or deterministic.

deterministic feedback

The distinction between deterministic and non-deterministic feedback has some consequences from the decoding point of view. Roughly speaking, deterministic feedback can be used to restrict the search space (hard constraints) whereas it is a little more difficult to deal with non-deterministic feedback since the system has to consider the probability of each possible hypothesis in order to integrate this information in the decoding process of the sequence to be transcribed (soft constraints).

In previous examples, user feedback and system response turns were discrete events and were interleaved, but other alternatives can also be considered, as is the case of "Speech Dasher" [Vertanen and MacKay 2010] where user feedback is in the form of continuous gestures while the system updates the view at the same time (no turns) and in real time.[34]

collaborative environments

Interactive transcription approaches should also consider additional issues such as the possibility of collaborative environments where several users may contribute to the transcription of a given corpus.

To summarize, interactive transcription systems take any new available information (mainly user feedback, and not only from the last turn) into account in order to:

- validate some part of the result, either explicitly or implicitly;
- update the information provided to the user relating the remaining part of the transcription. Note that this information is not

34 It is based on "dasher", briefly described in Section 2.6, see Figure 9.

only limited to the transcription of the remaining parts themselves, but can also include additional information such as confidence measures in the form of visual feedback, information relating the alignment between the transcription and the signal to be transcribed or even an improved presentation of the signal;[35]

- depending on the user model, the system can also adopt an active role and may be able to ask the user for some particular information. The system is not only limited to ask for particular errors, but can also formulate general questions about the nature of the task to select the most appropriate language model. The system can also propose the following sample to be transcribed in order to ask first for the correctness of labels the system is less confident. On this matter, a careful balance between system flexibility and ergonomics has to be considered, since a too flexible system may ask the user to change her/his context too often and this require an additional effort to be taken into account. These strategies may be useful in cases where the overall user effort has to be limited in advance so that a complete supervision of the entire transcription cannot be performed [Sanchez-Cortina *et al.* 2012; Serrano *et al.* 2013];

user adaptation

- improve the usability of the system by means of user adaptation techniques. Two different scenarios can be distinguished: casual users and long term (usually expert) ones. Adaptation makes sense in the last case. Different users may have diverse preferences and the system can adapt the profile automatically. Some input modalities such as online HTR or ASR can be improved by adapting their models to the user voice or handwriting style;

- improve the recognition capabilities of the system by retraining or adapting some models with new additional information, which is closely related to *bootstrapping* techniques: recognition systems are trained with a subset of samples and, once the interactive system was better than a pure manual transcription system, it can be used to produce even more labeled data in order to improve the models. It is even possible to use the remaining unlabeled training material with semi-supervised training methods and to rely on active learning techniques to find out which samples should be transcribed first to reduce the overall user effort. Model adaptation techniques can also be used to take profit of models from other corpora.

The last two points may not reduce the post-editing effort of the current transcription but rather get an amortized effect over subsequent transcriptions, so they make more sense in the long term.

As observed in Figure 32, some input modalities are redundant and this has some advantages: this not only allows different users to adapt to the system (the user would use a subset of the system capabilities depending on her/his preferences) but also allows the same user to change the input modality depending on at least two different reasons:

35 For example, some preprocessing techniques for handwritten text described in Chapter 14 can be applied not only for HTR but also to offer the user an easier to read view.

- to adapt the use of different input modalities to the particular features of the current sample. For example, it is probably better to correct the mistakes of a nearly correct transcription by means of an electronic pen but, when most words are incorrect, a keyboard may be more reasonable;

- to change the user posture and input modality can be useful in some cases to reduce physical and cognitive fatigue.

3.3 LIMITATIONS OF THIS CLASSIFICATION, HOW TO FACE THEM AND A NEW PROPOSAL

A great part of this chapter is devoted to review and enumerate some common problem types from the point of view of what is considered an input and an output. A summary is depicted in Table 2.

We have also seen how, in some cases, a problem from a given type can be reduced to another one from a different type. The approach consisting in providing an enumeration of problem types has, nevertheless, some limitations. On the one side, it is not easy to specify a fine grained hierarchy since several problems considered as different are not distinguishable from the *type* of input and output variables. For example, "verification" is the task of determining if a given pattern (utterance in speaker verification, although there also exist handwriter identification and many other verification tasks) corresponds to a given speaker/writer. This task could be considered, from a pure input output point of view, as a particular case of segment classification, although the techniques required to solve this problem are not the same as those used for other segment classification tasks. In a similar way, speaker diarization [Tranter and Reynolds 2006] consist in dividing or partitioning an audio stream into segments according to the sources associated with these segments. These sources comprise background noises, music, and the voice of different speakers. This task entails dividing the recording into homogeneous segments (speaker segmentation) and determining when different segments correspond to the same speaker (speaker clustering). Sometimes, additional information such as the number of speakers, training material for each of them, or even the transcriptions, are also provided. Diarization, again from a formal analysis of the type of inputs and outputs, seems very similar to joint segmentation and classification despite the use of specific techniques.

verification
diarization

Who said when?

As discussed at the beginning of the chapter it is common, in the statistical approach to pattern recognition, to describe a problem by means of set of random variables and the relationship between them by means of graphical models, which will be addressed in following chapter. When problems are specified in terms of inputs and outputs, the same model might correspond to different problems depending on which variables are hidden and which ones are observed which cannot always be established a priori, at least in some complex tasks.

Some examples and problem types described so far showed the idea that the concept of input and output may depend on the point of view

Problem	Input	Output
fixed dimension classification	fixed dimension features	class label
sequence classification	sequence	class label
segment classification	segment within a sequence	class label
fixed dimension joint classification	group of fixed dimension features	group of the same size with a class label per element of the group
sequence labeling	sequence	sequence (of the same size) of class labels
joint classification of segmented sequence	segmented sequence	sequence of class labels of size the number of segments
segmentation	sequence	the input sequence segmented
over-segmentation	sequence	several segmentations of the input sequence
recognition	sequence	boolean value (accept/reject decision)
parsing	sequence	data structure efficiently representing sets of compatible structures (parse trees)
sequence modeling	sequence	a priori probability of generating the sequence
prediction	prefix of sequence, word	probability of word extending prefix
translation	sequence in source language	sequence in target language conveying the same meaning
parallel text alignment	bitext	the same bitext aligned
transcription alignment	sequence and its corresponding annotation	annotated sequence
sequence alignment	(usually two) sequences	aligned sequences
keyword spotting	sequence and keyword	positions of keyword in sequence
query by example	two sequences	positions where the keyword described by first sequence appears in the second
joint segmentation classification	sequence	most probable sequence annotations
summarization	sequence	shorter sequence conveying the same meaning
rate (of speech,...)	sequence (signal generated from a hidden sequence)	estimation of the ratio between the observed and the hidden sequence lengths
filtering	noisy signal produced by a state-space model up to time t	the state distributions at a position t
smoothing	entire noisy signal produced by a state-space model	the state distributions at a desired position t
source separation	several signals mixed together	one of the signals

Table 2: Problem hierarchy described in terms of inputs and outputs.

and on the description level we are talking about, since what is the output of a sub-problem may be part of the input of another one.[36] Let us consider the case of multi-pass systems where the outputs of an stage constitute the inputs to the next one. The case of assisted transcription is even more subtle: the same *stage* can be iteratively applied in successive refinements and the number of steps is not known a priori.

We have also seen that many different types of information can be provided to extend most problems: imperfect transcriptions, segmentation constraints, validated prefixes, the presence of corrupted parts of the signal or auxiliary features, to mention a few. Even if many complex tasks can be composed into sub-tasks and some of them may fit in one of the cases described in previous sections, the variability of problems is so broad that the idea of making a large enumeration of types does not seem the best approach. Hence, instead of providing a hierarchy of problems of growing complexity described in terms of inputs and outputs, we believe that it could be modeled otherwise.

This section is devoted to discuss and to provide some first steps or to point out some ideas into this direction. It has been inspired by a linguistic annotation formalism known as *annotation graphs* proposed by [Bird and Liberman 2000]. Linguistic annotation is a general term which covers descriptive or analytic notations applied to language data, although it can be generally applied to other types of sequences and even to other type of structured data. Annotation graphs are proposed as a formal framework for linguistic annotation with a focus on *what* can be expressed no matter the particular implementation details such as formats to store them or particular tools to manipulate them.

The problem view as partial annotation

Roughly speaking, annotation graphs consider the input signals as an immutable space where some regions or segments are identified and labeled. Different types of annotation may coexist in the same representation, some parts may constitute a hierarchy and others a flat representation. It is also possible to specify gaps as well as overlapped segments. Annotation graphs are formalized as labeled acyclic graphs attached to the observed signals. Nodes of these graphs are associated to elements from some ordered set (which are termed as *timelines*) and may be related to sequence positions, although some nodes may not be attached to a precise sequence position. Arcs are labeled and represent segments. Paths where internal nodes are not associated to sequence positions can be used to specify label sequences related to a given segment with no additional segmentation. In this way it is possible, for instance, to relate a sequence of phonemes and a word to a given segment without specifying its phonetic segmentation. Independent annotations can be represented by disposing of several nodes or arcs associated to the same sequence position. A corpus is a set of annotated sequences so that it can be considered as a straightforward extension of sequences (by considering independent graphs as a sole graph) and it is also possible to operate on them by means of union, intersection and other algebraic operations.

annotation graphs

36 A different example which *seems* to violate the division between input and output is filtering and fixed lag smoothing, where the output is an estimate of the input, but it suffices to consider the filtered input as a different output random variable.

From this description, annotation graphs can be considered a quite general way to represent the input and the result or the output for many different problems described so far. Nevertheless, we believe that they lack some properties to fit our requirements: First of all, we would like to be able to specify the concept of *online* recognition, meaning that computations may be performed while the input signal is being generated. To this end, observation and annotation stages may be interleaved and the signal, although immutable, is no longer an static space given a priori.

persistent representation

Relating immutability, the entire process can be considered the construction of a persistent representation where observation and annotator agents are reified and explicitly represented into the formalism.

The construction of this representation leads to a partial order where some parts may be required first. To illustrate this idea, let us remember the concept of bootstrapping:[37] models used by automatic annotators are trained with a subset of samples and the automatic annotation produced in this way is used to easy the transcription of more data which, in turn, is used to estimate improved models. There exist a relationship between a particular model and the data which has been used to estimate it.

Such a representation could deal with a variety of concepts such as:

- online systems, by allowing the specification of the observed data as the incremental[38] construction of sequences, although it is much more general to extend the idea of sequences to the concept of DAGs and even to extend them to obtain a formalism roughly similar to packed shared forests;
- the difference between indexing and search times in KWS or STD systems;
- interactive transcription systems, even in a collaborative framework (several users are working on the same corpus), capable of performing user adaptation, bootstrapping, and active learning techniques to reduce the overall effort.

traceability and reproducibility

The concept of a persistent representation constructed in an incremental way by several agents perfectly fits in the idea of distributed *revision control systems*[39] and its interest is not limited to the task of classifying a particular pattern but, rather, to provide a way to easy the management of the overall process which also involves the acquisition and annotation of a corpus, the estimation of successive models required in the process and the possibility of a perfect traceability to be able to know which annotator, either automatic or human, has provided which information in which context at every moment.[40] Indeed,

37 Remember also the pool-based active learning cycle [Settles 2010].

38 The idea of producing a DAG incrementally is discussed in Chapter 8

39 See `http://en.wikipedia.org/wiki/Revision_control`. In particular, distributed revision control systems also take into account the idea of different "committers", the possibility of maintaining several branches at the same time, which can be merged later, and the existence of a partial order which relates the different actions. In our case, the role of committers is shared between human users and automatic systems.

40 Readers with some practical experience in the design and the acquisition of datasets, or who has worked with a huge quantity of models, annotators, and different versions of

this traceability moves in the direction of improving the reproducibility of the experiments [Sandve *et al.* 2013].

We believe that these concepts are an interesting point of departure to tackle more complex problems such as the extension of these ideas to higher dimensional data, as is the case of document analysis (which includes layout analysis as a particular case).

3.4 QUESTIONNAIRE WHEN FACED WITH A NEW PROBLEM

In the introductory chapter we shown the interest in trying to trace a map on the space of problems and another one on the space of techniques so that we could relate them by providing some criteria to determine which models and techniques are more suitable for a given task under some particular conditions. Since we have not discussed problem techniques yet, it is premature to try to reason about the properties required to establish a relationship between problem types and methods to solve them. Nevertheless, given the previous examples and the list of problems, we can make the following mind experiment: *mind experiment*

> *Facing a new problem related to sequences. Which questions would best help to locate the most suitable problem type associated to it?*

We believe that these questions should be taken into account when trying to study problem solving techniques in a task independent way. The following questionnaire is an attempt to address those questions:

- is it possible to describe the problem in terms of a "fundamentally one-dimensional" signal? Let us remember the case of HTR where a text line image is bi-dimensional but the complexity relating one of the dimensions can be bounded. The same can be said about sign language recognition where the body configuration can be described by means of a fixed number of parameters and where those body configurations over time constitute a sequence;
- is it a continuous or a discrete process? In case it is a continuous one, is it possible to obtain an accurate approximation by means of a discrete signal?
- is the signal being produced at the time we are processing it, or we dispose of the entire signal from the very beginning? This will affect the type of performance measures we should try to improve, and also the possibility of trying to use incremental recognition systems;
- do we need to provide feedback to the user as the signal is being produced? This is the case of some predictive text entry methods;
- can the system benefit from previous recognitions? Interactive assisted transcription methods allow the user to correct the output

experiments can perfectly understand the interest in these more pragmatic but less pure scientific objectives.

of the recognition so that the system should produce a new transcription taking into account the available information as much as possible. The capability of the recognition system to reuse information computed in previous iterations can be very relevant to improve the performance;

- can the observed signal be properly described as a concatenation of segments or can they be overlapped? These hypotheses are crucially related with the type of models we are investigating. In particular, our study does not deal with overlapped segments but rather on the "beads-on-a-string" paradigm;
- are there ambiguities when determining segment boundaries? For instance, it is not always clear in speech where a phone ends and the next starts due to co-articulation;
- does the task specify a clear repertoire of types of segments? It is not always clear which is the best repertoire of such units, as exemplified by the large repertoire of proposals for the case of ASR described in Section 2.8;
- are those segments made from another repertoire of smaller pieces? In that case, can we think about a lexicon? Can this lexicon be established a priori? Unconstrained systems have to deal with the problem of out of vocabulary words (OOV) and this has to be taken into account by decoders;
- is it possible to obtain an accurate segmentation by "external" means or this segmentation is best explained as a byproduct of a complete joint segmentation and classification process? A segmentation of speech utterances into phones by means of spectral changes is not reliable. However, it seems feasible to provide a reliable segmentation of handwritten lines into words, at least for some scripts, analyzing the space between connected components. The possibility of external segmentation can not only increase the decoding efficiency but can also improve the quality of results since some wrong hypotheses are not considered;
- are segments determined by intrinsic properties or are they considered segments because of its relationship with neighboring ones? For example, paragraph segmentation [Shi *et al.* 2007] is usually described in terms of some type of *internal coherence* measures which depend on features obtained with respect to other paragraphs;[41]
- can dissimilitude measures (such as distances) be useful to classify segments? That is, may closer segments more probably belong to the same type of label?
- which is the probability that random segmentations of the signal are similar to correct segments? Some approaches such as the "antiphone" described in Section 4.4.2 implicitly rely on a low

41 Let us remark that we are not interested in this type of problems, we limit our study to problems where segments can be thought as generated by a stochastic process given a type of hidden or underlying unit or label. But note that, even if segments can be explained as the result of a generative process, it is possible, nevertheless, that they are related to neighboring ones. For instance, in ASR there are tonal languages where tone is relative instead of absolute.

probability of this fact. Note that a random segmentation of a handwritten line can lead to correct graphemes, at least when these considered isolatedly (without considering the surrounding context), which do not correspond to the correct transcription. For instance: left part of letter "d" is similar to letter "c";

- are similar segments to be considered as different only by their respective length? This was the case of Finish language where some words are distinguished by the duration of some vowels. This has some consequences on the choice of segment models and decoders which should take into account duration;[42]
- can the classification of segments proceed in a hierarchical way? Speech sounds can be classified as voiced or unvoiced;
- what can be deduced about the nature of a segment given just one frame? For instance, it seems easier to perform *"frame-wise phoneme classification"* in ASR than the corresponding "frame-wise grapheme classification" HTR counterpart.[43]

3.5 ANALYSIS AND EVALUATION MEASURES

measures are purpose oriented

The evaluation of pattern recognition systems has several purposes: to compare different techniques, to choose the most appropriate one, to analyze what can be improved or as a criteria to train models. They can measure how well the result is adapted to the user wishes, the performance, etc.

with and without ground truth

Evaluation measures usually compare the output produced by the system with the corresponding ground truth or correct reference.[44] A clear example is the task of classification, where the error rate can be estimated as the ratio of correctly classified patterns. This value is a particular case of a more general scenario when the loss incurred by a misclassification may vary depending on the action taken by the classifier and the true state of nature. Loss functions are related to machine learning criteria described in the following chapter.

loss function

precision & recall

In other tasks, such as KWS or information retrieval, we are also interested in measuring the relevance, which is associated to the concepts of precision (the fraction of retrieved samples considered to be relevant) and recall (the fraction of relevant samples that are retrieved).

When computing assessment measures, it is often reported the expected average case. Unfortunately, the true probability of the space of test samples is not known and these estimations have to be approximated empirically over an independent test set. That is why confidence intervals are usually reported to determine how reliable are the results and to allow the study of statistical significance.

42 These issues will be addressed in Chapter 7.

43 Note that we are not calling the idea of frame-wise classification into question (since it can always be obtained as a byproduct of the overall process by marginalizing over all word posteriors overlapping a given frame) but, rather, taking about using only the frame to obtain this information.

44 Nevertheless, the automatic evaluation of systems, when a ground truth or gold standard is not available, is also possible.

Note also that the design of formal evaluation methodologies include the design of relevant benchmarks. The material of these tests has to be representative in most cases, but it may be specially designed to analyze some particular feature in other cases. Developing reliable and appropriate tests is a discipline by itself.

confidence measures

Some classifiers are can also provide indications about how much we can trust their results. For example, some ASR systems can produce, in addition to the transcription, a score[45] known as *confidence measure* to indicate the reliability of the output. Confidence measures are useful, for instance, to implement the *reject option*, which is the possibility of some systems to prefer to refuse to take a decision rather than taking a wrong one.

several references

In some tasks, as is the case of translation, it is possible to provide several reasonable references and this possibility should be taken into account when designing the corresponding evaluation method.

analysis

It is also important to be able to analyze and to compare the systems in order to understand their weakness and to obtain some insight to propose new ways to improve them. To this end, a sole number or statistical measure does not usually suffice. These measures can be obtained for different versions of the system by varying some parameters in order to understand their effect, but it is also convenient to extract data, not directly related with evaluation measures, with the sole purpose of analysis. For instance, we could retrieve the best representative errors in order to try to understand the reasons why the system produced them.

performance

In addition to assessment measures, we are also concerned with the performance or the computational costs required to solve the tasks. Indeed, assessment and performance have to be considered at the same time since there exist a trade-off between them, usually in the form of the law of diminishing returns[46] and a compromise has to be adopted.

user effort

The case of interactive pattern recognition systems deserves special attention relating the measure of the average user effort normalized by the size of the task to be performed, since it is not obvious at all what has to be measured and how to measure it without a simulation with real users. A subsection is devoted to discuss this issue.

complementarity

Besides assessment, performance or effort, there exists other less obvious types of measure such as the complementarity, concision, robustness or flexibility. The complementarity between different systems, related with the correlation between the errors produced for the same inputs, seems a useful measure to determine the gains that could be obtained by combining the systems.

concision

Concision and coverage are concepts commonly applied to certain output formats such as word graphs. Perhaps, one of the measures more difficult to formalize are those aimed at measuring the robustness or the flexibility of a given system. These measures would address what happens when the task at hand changes some conditions or how easy would be to adapt the system to new

robustness flexibility

45 Posterior probabilities at the word level are a good example of these measures, although other more or less heuristic measurements have also been proposed.

46 The law of diminishing returns, described in the introductory chapter, applies here: once we are approaching the best results, more and more sophisticated and resource intensive methods are usually required to barely improve the results and we have to decide which computational resources we are willing to invest.

tasks. The unexpected nature of possible future changes makes us cautious about them.

intrinsic & extrinsic

In many cases, the task we are trying to measure is part of a bigger one and the evaluation, no matter if we are interested in assessment, relevance or performance measures, can be performed in two different ways:

INTRINSIC when we are directly measuring the value of interest;
EXTRINSIC consists in measuring the output of the overall end application in order to evaluate the contribution of a part.

An example of extrinsic measure is the case of text alignment, which makes usually part of machine translation systems. In this case, we could measure the alignment quality in an intrinsic way (described below), but also in an *indirect way* by studying the difference in translation quality when different alignments are applied to the same SMT system. This approach has pros and cons: on the one side, we are usually measuring what we are really interested in, but, on the other side, we have to be very cautious since this measure also depends on the rest of the system and it is not always clear the interaction with other parts or the effect or contribution of the part we are interested in.

objective & subjective

Another distinction when classifying evaluation measures is in terms of objective and subjective measures. Subjective measures are more costly to obtain and are subject to user biases and inter-rater disagreements. They are more appropriate than objective measures in some particular tasks or evaluation purposes. Objective measures, on the other side, do not exhibit those problems but we do not have to forget that their usefulness comes from the correlation with their subjective counterparts.[47]

This section is an sketch which could be greater improved. It is not possible to grasp the entire complexity in a few pages. Indeed, the interested reader will be referred to appropriate bibliographic references covering particular aspects into more detail. Let us finish the introduction to this section with two little remarks related to evaluation of systems:

- it is important to acknowledge how delicate it is to fairly compare different systems experimental results. On the one side, minimal variations of conditions and parameters (more probable when performed by different authors) may cast doubts on the validity of the comparison. On the other side, even if the comparison is performed with the same settings, the validity can be questioned because of implementation bias (specially when performed by the same authors who are often proposing one of the systems as better).

  A necessary step to improve this scenario would be the to assure the reproducibility of the experiments as suggested by some authors [Sandve *et al.* 2013];

- the increase of accuracy of a system does not necessarily mean that we are performing a true progress in the research field. Some

47 For instance, WER (defined below) seems more correlated with the subjective impression of readers than SER. Interactive systems are more prone to be evaluated by means of subjective measures.

improvements may be due to the fine tuning or thanks to the increase in the training data size or the because more powerful computers allow the use of models which were prohibitive before. Although this is important and can be justified, we cannot forget that sometimes advances can be accompanied of the increasing of recognition rates [Bourlard *et al.* 1996].

The rest of this section is structured in terms of the different types of measures.

3.5.1 Quality assessment

loss function

In order to assess the behavior of classifiers in terms of the quality of the output, it is usually supposed that a non-negative real-valued *loss function* $\lambda(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ can be provided to measure the loss[48] incurred in choosing an action a for a given true state of nature s. Some loss functions are commonly used to evaluate classification tasks:

KRONECKER, or 0-1 loss, leads to the concept of classification error. When this loss is applied to problems which produce an output sequence, as is the case of the joint segmentation and classification task, we obtain what is known as "Sentence Error Rate" (SER). SER considers the sentence as a sole label/entity and, consequently, does not take into account the number of mistakes in a not perfectly recognized sentence. Put in other words: just a little error has the same impact on the measure as a completely wrong transcription;

HAMMING is one of the most common loss functions proposed for joint classification (sequence labeling) tasks where the output of the system is a fixed length sequence. This function measures the label-by-label loss or number of incorrectly predicted labels. The output and the reference have always the same length and the concept of substitution suffices to convert the output into the reference;

LEVENSHTEIN distance is one of the most common edit distance or string metric to compare two different sequences of possibly different length, as is the case of the joint segmentation and classification tasks (ASR, HTR,...). This distance is the minimum number of substitutions (S), deletions (D) and insertions (I) required to transform the output of the system into the reference sequence. The rationale behind this measure is that it roughly measures the effort required to correct the output transcription. When it is applied at the word level,[49] this measure is known as

48 So the lower the loss the better, in case of approximation problems this value is also known as discrepancy. Other works propose the use of utility functions but both approaches are equivalent by negating the utility function to get a loss. These concepts will be described into more detail in Section 4.1.

49 Besides words, this evaluation measure can also be applied to any type of tokens or lexical units whatsoever. So, for example, if we need to measure the accuracy of HTR systems character-wise, it suffices to compute the Levenshtein distance at the character level. In this case, the value is usually known as "character error rate" CER (this acronym collides with "Concept Error Rate").

"Word Error Rate" (WER). WER is normalized by the length of the reference word sequence in order to easy the comparison between different systems on different tasks, as the magnitude of the distance depends on the length and is defined as follows:[50]

$$\mathrm{WER} = 100\frac{\mathrm{S}+\mathrm{I}+\mathrm{D}}{\mathrm{N}_{\mathrm{ref}}}$$

where $\mathrm{N}_{\mathrm{ref}} = \mathrm{S}+\mathrm{D}+\mathrm{C}$ corresponds to the number of words in the reference transcription (C the number of correct words). Note that this value is not a percentage and values higher than 100 are perfectly possible.[51]

In order to properly compare different systems, we have to be aware of the fact that two systems producing the same statistical values[52] may produce errors in different samples and, conversely, that the same system may produce different results when applied to different test sets so that an increment or a decrement in WER, for instance, when applied to a relatively small dataset, may not imply a real improvement in the system but may be due to chance. It is therefore necessary to use significance tests to compare systems [Gillick and Cox 1989] or to provide not only a figure of merit but also a confidence interval for it. A method for computing these intervals for WER without simulations is proposed in [Vilar 2008]. Some authors consider interesting to dissect the WER into which parts are due to OOV words in order to obtain a lower bound to better place the reported WER.

In some cases, the error is reported at the frame level even for joint segmentation and classification tasks (ASR, HTR,...). This is the case of the "frame error rate" (FER) concept described in [Wessel *et al.* 2001; Sect. 2.1] which only takes into account substitutions at the frame level.

Quality assessment of SMT

Statistical Machine Translation (SMT) has been briefly addressed in Sections 2.4 and 3.1.9. This task is more complex than the joint segmentation and classification of sequences due to reorderings. Although the suitability of models studied in this work for this task seems limited to the monotone case, we have to remark that some of the tools used in SMT are appropriate for our study, as is the case of language models (LMs). This is one of the reasons why we can justify to talk now about evaluation measures for SMT since some contributions on language modeling can be measured in SMT tasks. Indeed, some experimentation with LMs presented in Chapter 13 makes use of these evaluation measures.

The evaluation of SMT systems (and even the evaluation of translations made by human experts) remains an open problem not exempt from controversial. Among other reasons to explain this difficulty, there may exist many possible translations for the same input. Both

50 Other similar measures such as "percent correct", "word accuracy", "global error rate" or "percent total" have also been proposed, the interested reader is referred to [Lee 1988; Prat *et al.* 1994; Hunt 1990].

51 For instance, with outputs arbitrarily long, the number of deletions may be very high.

52 Excepting some trivial cases where the output is perfect.

human and automatic evaluation measures can be used. The following concepts can be considered for human judgment:

ADEQUACY measures to what extent the output of the SMT system has the same meaning that their input (or, alternatively, than the translation references);

FLUENCY measures if the SMT output is fluent (e.g. using the word choices a native speaker would have used) without taking into account if the meaning is correct.

Relating automatic assessment measures, several ones have been proposed in the literature such as: the word error rate (WER), the BLEU (BiLingual Evaluation Understudy), the NIST, the "translation edit rate" (TER), etc. Most of them aim to model the correspondence between the output produced by the SMT system and a reference (or set of possible references, in some cases).

Let us now see into more detail the two of the most widely spread evaluation measures. They will be used in Chapter 13.

BLEU (BiLingual Evaluation Understudy), introduced in [Papineni *et al.* 2002], is based on the geometric average of a modified n-gram precision (for n-gram orders from 1 up to N which is set to $N = 4$). Unlike WER, BLEU is clearly a measure where the higher the value the better. The n-gram precision is related to the number of n-grams shared by the SMT output and the reference. Since the result of a geometric mean is zero when one of the counts is zero, this precision is evaluated on the entire test corpus, and it is computed as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

where BP is a brevity penalty defined as:

$$\text{BP} = \begin{cases} 1, & \text{if } c > r \\ e^{(1-r/c)} & \text{otherwise} \end{cases}$$

being r and c the length of the reference and SMT output, respectively. The other elements to define BLEU are w_n and p_n: w_n are positive weights summing 1 that are set to $1/N$, while

$$p_n = \frac{\sum_{C \in Candidates} \sum_{w \in C} Count_{clip}(n\text{-}gram)}{\sum_{C \in Candidates} \sum_{w \in C} Count(n\text{-gram})}$$

is a modified precision score based on the ratio between counts with and without clipping. Clipping means limiting the count of each candidate n-gram to the maximum number of times appearing in a single reference sentence.

TER Evaluation measures may be purpose oriented and some of them, as is the case of "translation edit rate" (TER) [Snover *et al.* 2006], try to evaluate the cost of a post-editing in order to correct the output of the SMT system (i.e. to make it identical to one of the

references). As with the word error rate, it is based on the Levenshtein distance that is extended to allow, (besides insertion, deletion and substitution of words) shifts of word sequences. Although these shifts make the exact computation of TER intractable, there are available implementations using heuristics to approximate the optimal value. The TER is defined as:

$$\text{TER} = \frac{\text{Number of edits}}{\text{Average number of reference words}}$$

so, as with WER, the lower the value the better.

3.5.2 Performance

Performance measures are related to particular decoder implementation details and system settings.[53]

The most common performance measure is the time required to decode an input sample, in average, normalized by the input length, which is a measure of throughput. Another other interesting measure is the latency which is related to the delay required to obtain a part of the output when some part of the input has been already available.

In some tasks where really huge sequences are common, as in some bioinformatic tasks, it is also interesting to measure the memory requirements and even to sacrifice time to reduce memory, but we will not cover evaluation of memory requirements here.

Throughput

Can be defined either as the time required to process an instance of the problem, either as the number of instances per unit time (in both cases, normalized by the instance size).

The most common measure of performance in ASR is the "Real Time Factor"[54] defined as:

$$\text{RTF} = \frac{\text{time required to recognize the sequence}}{\text{length of input sequence (in time)}}$$

RTF is important when processing a continuous stream: a RTF greater than one would mean that the part of the sequence to be recognized will grow faster than the part already recognized.

In other tasks, such as offline HTR, the length of the input signal is not the time, and the concept corresponding to RTF would be the time required to decode a pixel column or a frame (or the inverse: frames decoded per second). Nevertheless, it is more common to report the throughput in terms of lines or words (e.g. words processed per second). The problem of this measure is that it is dependent on factors such as the average word length or the image resolution.

53 The performance of a system may depend on the specific characteristics of the computer platform, remember the successful trilogy from [Aubert 2002] described in Section 1.2.

54 Unfortunately, there exist other nomenclature and some authors also report results using "times faster than real time (xfRT)", which is the inverse of RTF.

Latency

Latency is the time we have to wait to obtain the output once the input is available. From the decoding point of view, this measure of performance makes sense in at least the following scenarios:

- although both concepts are not necessarily opposed, do we prioritize throughput or latency?
- in some tasks the input signal can be processed while it is being produced. Some recognizers, known as "online",[55] are able to start decoding instead of waiting for the entire signal to be at their disposal;
- a concept somewhat related to "online recognition" is "incremental recognition" which is the capability of producing partial outputs before the entire decoding process has finished. Some incremental decoders are restricted to emit the correct partial output whereas others are allowed to correct previous partial outputs when more information is available. The first ones usually emit the longest prefix which is compatible with all remaining active hypotheses.
  It is important to remark that latency, in the context of incremental recognition systems, is not only a measure of performance, since there exist limitations inherent to the task at hand *no matter the decoder speed*. For instance, in ASR it is usually necessary to wait for a right context in order to assure the presence of a word being transcribed.

3.5.3 Interaction

The most valuable resources to measure the cost of an interactive transcription system is associated to the user: time and ergonomics (cognitive and physical effort). Some authors propose to measure the expected number of user corrections, normalized by the size of the transcription. For instance, [Toselli *et al.* 2008], within the framework of interactive transcription of handwritten documents by means of online strokes (gestures and online HTR), defines the "word stroke ratio" (WSR) as the number of (word level) user interactions required to obtain the reference transcription of the text image considered, normalized by the total number of reference words. A similar measure, known as "key strokes per character", can be found in the field of predictive text input methods (Section 2.6).

These expectations can be obtained by means of simulations, but this poses a lot of problems excepting some particular scenarios where the user behaviors can also be simulated by a system.

It is important to realize that some user corrections are more costly than others. For instance, deleting a word by means of a gesture on the screen can be faster than rewriting an entire word. Sometimes, writing the word using online HTR is slower than using the keyboard, but there is a cost when changing from an electronic pen to the keyboard, so the cost may depend on previous user actions.

55 Do not confuse with "online" in the context of "online HTR".

The normalized expected number of corrections should be weighted by the effort of each individual correction taken into context although, in practice, great simplifications are assumed and all corrections are usually assigned the same cost. The problem of properly assigning a weight to each different user action is not trivial, although there exit some studies on the human movement in the framework of ergonomics and human-computer interaction such as the Fitt's law.

Expected number of user corrections also ignores the cost of supervising the correct parts of the transcription, hence ignoring the time and the cognitive effort made by the user. For instance, an interactive system which proceeds in a left-to-right order could update the suffix to be validated after each user feedback but, even if the latency is greatly reduced, the user may have invested some effort perceiving a part that is constantly changed. Even worse, a system which actively ask the user in what he or she perceives as "random order" could reduce the expected number of user corrections but probably at the expense of a greater user nuisance. As we have pointed out before, the field of assisted interactive transcription could benefit from some useful ideas from the field of (spoken) dialogue systems in order to identify and adapt the system to user expectations.

The time required to correct the transcription and the time response (latency) at each interaction are also usually ignored in most words when reporting results in the field of interactive assisted transcription.[56] It is difficult to measure the time response: the expected time is not very informative if variance if high and user perception of delay is probably far from a linear loss function.

We are not only very critical with the limitations of current measures but also very cautious with the methodological problems of measuring the expected cognitive and time effort of users. On the one side, it is very difficult, even impossible, to simulate user actions excepting some very simplistic scenarios. On the other side, results should be parametrized by a lot of particular, sometimes subjective, features. These measures should take into account user preferences on ergonomics, which may depend on the target user. We should distinguish the case of an expert user whose behavior can be studied after the system has been adapted to him/her and the case of casual users which should be classified, in the measures, depending on their preferences. We also consider very important to distinguish the measures of user effort on the long term usage since it is not at all the same to use a system five minutes a day or eight hours a day, every day.

This area could benefit from a collaboration with areas such as psychology, psycho-metrics, design of experiments, or ergonomics. As pointed out in Section 3.5.3, in the ideal case the system should anticipate the user intentions by taking into account the use of eye tracking devices, web-cams to analyze unconscious gestures and with sophisticated[57] user models as if the system was "inside users' mind".

56 Authors probably assume they are not relevant because they are working with "prototypes" which could be further improved.

57 As pointed out by Arthur C. Clarke: "Any sufficiently advanced technology is indistinguishable from magic".

3.5.4 Other measures

Besides the assessment of recognition systems, the study of their performance and the case of interactive systems, other different types of measures and measures for other tasks can also be envisaged. Although the following measures probably deserve a section by themselves, we will summarize them for the sake of brevity:

INCREMENTAL SYSTEMS are those able to process the input while it is being produced and who are capable of start generating an output that can, consequently, be used by other tasks. There exists some evaluation metrics specially designed for incremental systems [Baumann *et al.* 2009].

CONFIDENCE MEASURES can be used in several problem types. They can be used to label individual words in the output of the recognition system as either correct or not, but they are also related to many information retrieval tasks such as keyword spotting where a subset of segment positions, documents, etc. are retrieved as the result of the user query. Several metrics are used in the literature to report confidence measure results, among them we can mention the normalized mutual information metric and the equal classification error rate [Siu and Gish 1999].

ALIGNMENT AND (OVER-)SEGMENTATION are tasks where the usual metrics include the alignment error rate, which is derived from the precision and recall measures [Och and Ney 2003; Eq. 42]. In this context, precision measures how many boundaries identified by the system are correct whereas recall measures how many boundaries have actually been detected. Among other measures we can mention the average value and standard deviation of the distance between the boundaries proposed by the system and the reference [Toselli *et al.* 2007b; Sect. 4].

COMPLEMENTARITY is a measure to compare the dissimilitude between different systems aimed at solving the same task. This measure could be used, for instance, to determine if two systems reporting the same average error are producing the same mistakes on individual samples. Complementarity seems useful to guess the expected gains that could be obtained by combining the systems. [Burget 2004; Chapter 3] proposes complementarity measures based on counting simultaneous and dependent errors.

3.6 SUMMARY AND SOME CONCLUSIONS

This chapter complements the previous one that was devoted to introduce some case studies. Both of them are dedicated to tasks and problem types related to sequences without explicitly addressing *how* to solve them.[58] In this way, they also complement the next chapter

58 Which may seem unconventional to some readers. Indeed, any description of many interrelated concepts constitutes a "chicken-and-egg" dilemma about how to serialize ideas. This particular procedure has probably some drawbacks, as illustrated by the fact that we have been tempted to talk about problem solving techniques.

devoted to describe and compare models for dealing with the joint segmentation and classification on sequences.

We have observed that it is not easy (or even possible) to provide a widely accepted and clear nomenclature for the set of common problem types related to sequences in a *task independent way*, not to mention the large number of new problem types which can be devised whose terminology has not been coined. The presence of nomenclature clashes and ambiguities implies that the meaning should be deduced from the context in which it appears. We have not claimed at all the proposal of any unified nomenclature,[59] although it is a pity that some concepts such as word lattice and word graph do not have a clear and standard definition.

We have tried to characterize problems in terms of their inputs and their respective outputs, showing how some problems can be decomposed into simpler ones, how they can be stacked or combined, or how they can be posed in terms of others, which does not necessarily imply a mathematical equivalence, since there may exists some different assumptions. The possibility of reducing one problem into another one may give the wrong impression that the first problem is easier or more restricted than the second one. Indeed, there may exists cyclic relationships![60] For instance, the problem of frame classification (e.g. to assign the probability of each phonetic label for each acoustic frame of a spoken utterance) may require to solve the joint segmentation and classification problem which seems much more difficult, but this last problem is usually based on a frame classification, an *apparent* circularity which does not hold.

Note also that the same problem type can be solved by means of quite different techniques. It is possible, for instance, to estimate[61] the rate of speech (phonemes per second) by means of ASR or by means of signal-processing metrics [Morgan and Fosler-Lussier 1998]. Both techniques try to answer roughly the same question, but the cost, reliability and, hence, applicability are different.

We have discussed some limitations to the input and output approach to characterize problem types and we have pointed out the suitability of graphical models[62] and the use of ideas inspired by annotation graphs in order to propose a way to express the complexity of interactive multi-modal and other complex systems. The proposed approach is based on reifying the different agents which participate in the complex annotation tasks (in a broad sense which includes acquisition, training and generalizes the particular case of transcriptions), including not only the training data but also human (oracles, but also acknowledging the possibility of errors and disagreement between different users) and automatic pattern recognition systems. Although trying to apply these ideas to some coined problem types would be overkill, we believe that the proposal of Section 3.3 clearly shows that

59 That is why we have avoided definitions in this part: it is not our purpose, we lack authority and this probably would make worse the situation, as depicted in Figure 5.

60 Should we talk, instead, of a basis or group the strongly connected components of this graphical relationship?

61 The rate of speech computed by means of ASR is also an estimation not only because ASR systems are fallible, but also because their purpose is to estimate the identity of words even if the user has not pronounced some phones.

62 Not to confuse with the graphical models as described in Section 4.1.1.

the state of the art in this field is far from being closed and points out a promising direction.[63]

The current chapter has not only described and organized some problem types but has also discussed different types of evaluation measures related to them.

Finally, this chapter has also started a questionnaire in order to help to situate a new problem relating the most suitable type and the most similar known task. We expect that some of these questions would help to anticipate some issues and requirements on decoders. Most decoders are based on assumptions which does not hold in the general case. We do not expect to generalize decoders in such a general problem-independent way in all cases but, at least, to make assumptions as much clear as possible in order to help other researchers to determine which techniques and which decoders are more suitable for a novel problem.

63 These ideas, together with the idea of tracing a map of problem types and techniques (and to relate them) are not only compatible but part of a same vision or plan which is out of the scope of this work.

4 MODELS

PRIMUS DOCTOR

Si mihi licenciam dat Dominus Praeses,
Et tanti docti Doctores,
Et assistantes illustres,
Très savanti Bacheliero,
Quem estimo et honoro,
Domandabo causam et rationem quare
Opium facit dormire.

BACHELIERUS

Mihi a docto Doctore
Domandatur causam et rationem quare
Opium facit dormire:
A quoi respondeo,
Quia est in eo
Virtus dormitiva,
Cujus est natura
Sensus assoupire.

Molière, Le Malade imaginaire

CONTENTS

Scientific models usually follow a reductionist approach, explaining things in terms of more elementary parts and hence avoiding circular reasoning like *"dormative powers of opium"* from Molière's satire. With this philosophy in mind, we tackle the problem of generating sequences by decomposing them into segments associated to a finite set of observation models. This doubly stochastic generative processes is not new: they include the widely known HMMs and also *some* segment models. An initial section is devoted to introduce some preliminary machine learning (ML) concepts to better understand the following sections. The "two stage" generative model is described in Section 4.2 which proposes a hierarchy for the second stage and devotes another subsection to discuss extensions, limitations and generalizations. This section is complemented with a short introduction to the three "classical problems" (namely, probability of observation, estimation and decoding). A final section is devoted to review some alternative models to better put the two stage generative model into context. This section address the relationship with dynamic graphical models and with segment models. A summary and some conclusions closes the chapter.

4.1 SOME PRELIMINARY ML CONCEPTS

Let us start with some preliminary concepts on statistical pattern recognition and machine learning (for more details we refer the user to reviews and books such as [Bishop 2006; Duda *et al.* 2001; Murphy 2012; Darwiche 2009; Barber 2012; Deng and Li 2013; Domingos 2012]).

rational decisions under uncertainty

Statistical pattern recognition is about making rational decisions under uncertainty. The use of probabilities is a coherent way to model and to reason about different types of uncertainty[1] of event occurrences which can arise for several reasons such as true physical random phenomena or due to an incomplete knowledge of the system, as is the case when another model is used to approximate our task. Problems are formalized by means of statistical models which describe the relationship between a set of random variables[2] which take on values with certain probabilities and which might or might not have the ability to influence each other. We will follow the convention of denoting random variables using upper-case letters and their instantiations in lower-case. The value of variables corresponding to observations is usually known, but there may also exist missing observations. Other types of ranom variables include the hidden or latent variables, distribution parameters, etc. The reasoning about the probabilities of one subset of random variables given values of another subset is called "probabilistic inference". Graphical models, described below, are a useful way to describe and to reason about the conditional independence (CI) of a fixed set of variables which may lack structure. The extension of these *ground* graphical models to the *dynamic* case (for temporal modeling, to provide a structure) using templates, plates, switching parents or other features will also be briefly described later in this chapter.

inference

finite data

Machine learning (ML) is interested in how these models can be estimated from a finite set of data, which corresponds to the inductive inference paradigm. Let us remark that ML is an area broader than what is described in this work since it not only covers the statistical and inductive inference point of views but also the use of techniques not necessarily based on probabilities (e.g. based on distances) and other approaches different from the inductive inference. For instance, the transductive inference is interested on using data to take decisions on some predefined test samples without generalizing necessarily to other patterns. This section describes some minimal ML concepts in order to provide some background to better follow next sections. It is focused on supervised learning and on classification, leaving aside other problems such as regression. Classification can be seen as a process to make decisions (or to take actions from a set $\mathcal{A}$) in terms of input features[3] (taken from $\mathcal{X}$). It can be studied from the *decision theory* point of view: a classifier can be seen as a "decision rule" or function[4] $\alpha : \mathcal{X} \mapsto \mathcal{A}$ which belongs to a space $\mathcal{H}$ of hypotheses (space of allowed

decision rule

1 There exist other approaches to uncertainty. For instance, ambiguity which might be tackled by fuzzy systems.

2 Most common types of random variables (including complex ones such as sequences and graphs, generally known as "structured data") have been discussed in Section 3.1.

3 Preprocessing and feature extraction has also been briefly discussed in Section 2.9.2.

4 For a given input pattern, it is reasonable to output always the same value.

decision rules, each model in the family is identified or indexed by a parameter θ) so a particular classifier α is specified from $\mathcal{H}$ given θ and produces an action $\alpha(x)$ given an observation x. Observe that there can be different possible hypothesis spaces, choosing one of them is related with model selection and is one of the assumptions taken by an inductive bias (defined below). An alterantive way of representing classifiers is by means of *discriminant functions*[5] which are functions $g_a : \mathcal{X} \mapsto \mathbb{R}$ assigned to an action a so that $\alpha(x) = \text{argmax}_{a \in \mathcal{A}} g_a(x)$.

model selection

In order to measure the performance of classifiers, it is usually supposed that a non-negative real-valued *loss function* $\lambda(a|s) : \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ can be provided to measure the loss[6] incurred in choosing the action $a \in \mathcal{A}$ when the true state of nature is $s \in \mathcal{S}$. In the case of classification and estimation, we are trying to guess the state of the nature, so we will assume, in the following, that $\mathcal{S} = \mathcal{A}$. Let us remark that although certain loss functions are often used with certain classifier types, they are not necessarily tied or paired and it is possible to use different loss functions with the same model and vice-versa. Also, as we will see, the concept of loss function also appears when training or estimating the model parameters. It is possible to train a model using a loss function with a criterion different from the one used at classification time.

loss function

Since true distributions are usually unknown, one of the goals of ML is to study sound ways of choosing a proper decision function given some empirical observations in the form of a finite set of training examples in order to obtain a good predictive performance (that is, a good expected behavior on unseen patterns). A good generalization usually entails avoiding to model coincidental or random features of the training data (i.e. to avoid *over-fitting*). Training patterns are usually assumed to be obtained in an independent and identically distributed (i.i.d) way, which corresponds to the passive learning setting. This is in contrast with active learning where the learning system is allowed to choose the data from which is learns. Also, depending on whether the samples are labeled or not we can distinguish between:

over-fitting

active vs passive learning

UNSUPERVISED when they are not labeled. This case seems more related to density estimation, dimensionality reduction or clustering than to classification. It is more related to descriptive than to predictive tasks. Some recent works try to estimate models for ASR and HTR tasks using unlabeled sequences assuming that both a lexicon and a language model are at our disposal, it is not clear if this corresponds to unsupervised or to a weak supervised case, the frontiers being perhaps subtle;

SUPERVISED when using labeled examples. Let us observe that, in the case of joint segmentation and classification tasks the patterns are signal sequences usually labeled at the word level without specifying the exact sequence of phonemes/graphemes nor the segmentation information. In this case we talk about partially supervised. This case, specially relevant from both decoding and training points of view, is the one addressed in this work;

5 Some authors use this term to functions directly mapping features onto actions.

6 Concept introduced in Section 3.5. The lower the loss the better. In approximation problems this value is also known as discrepancy. Some works propose the use of utility functions. Both approaches are equivalent since a negated utility is a loss.

SEMI SUPERVISED is the case when both labeled and unlabeled samples are at our disposal. This is one of the most common cases, in practice, for many tasks such as HTR or ASR where there may exists more samples than those that can be transcribed, due to the transcription cost. A fact that also explains the interest in active learning where one of the approaches consists of making the trainer to choose which samples a (human) oracle will transcribe. In any case, these transcriptions can be eased with the use of interactive transcription systems.

data is not enough

A finite set of data may be explained by several models, but a unique solution is usually required.[7] In this way, ML is ill-posed since the data by itself is not sufficient to determine a unique solution. It is therefore necessary to incorporate additional knowledge in the form of a space of hypotheses together with a prescription to choose an estimate from it. The set of assumptions is generally known as *inductive bias*. For instance, there are uncountable functions which give the same values for a finite set of points but, if we restrict the estimation to the set of linear functions, a unique solution is found[8] when we fit the points using minimum square error criterion. Examples of inductive bias are maximum margin (which try to maximize the width of the boundary between classes) or minimum description length (which try to select the hypotheses with a shorter description, related with the Occam's razor). The general prescription which tell us what to do with the data is known as *inductive principle* whereas the specific method to obtain an estimate is known as learning method.

To sum up, the following main ingredients are required by supervised ML on the passive learning setting for classification:

- an space of decision rules (functions to choose an action given the observation);
- a loss function to quantify the consequences of decisions;
- a family of statistical models (each model in the family is identified or indexed by a parameter θ);
- some labeled training data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ which is assumed to have the same distribution as the test;[9]
- a learning method (a particular implementation of an inductive principle), which is a mapping:

$$\bigcup_{n \geqslant 0} (\mathcal{X} \times \mathcal{A})^n \mapsto \mathcal{H}$$

Let us make a little digression in order to refresh some concepts, related to probability theory and to Bayesian updating, required to pursue the study on how to choose actions given observed evidences.

7 The Bayesian approach, described below, consider the possibility of taking into account the predictions of *all hypotheses* weighted by their posterior probabilities, but this does not affect the current reasoning. The Bayesian approach includes assumptions in the form of priors.

8 At least two points are required.

9 Unfortunately, this is not always the case. Indeed, even the tiniest mismatch between training and test conditions is one of the reasons of a decrease in performance and is one of the reasons for model adaptation.

Some probability theory concepts

- **Experiment:** phenomenon where outcomes are uncertain.
- **Sample space:** set of all possible outcomes.
- **Event:** a collection (subset) of outcomes, the collection of events being closed under complements and countable unions. A real-valued **random variable** is a mapping from the sample space to the reals so that the preimage of any range of values defines a valid event.
- **Probability:** a value between 0 and 1 associated to an event, it can be interpreted as:
  - **Frequentist interpretation:** (or sampling theory) the limit of an event's relative frequency in a large number of trials,
  - **Bayesian interpretation:** a degree of belief in the occurrence of this event.

  If an event will always occur, its probability is 1, if an event will never occur, its probability is 0. Sample space, events and probability define a probability space where the probability function satisfies the following properties:
- **Sum rule:** probability of a disjunction of two events A and B:[10]

$$p(A \vee B) = p(A) + p(B) - p(A \wedge B)$$

- **Conditional probability:** $p(A|B)$ is the probability of one event A given that another event B occurred.
- **Product rule:** probability of a conjunction of two events:

$$p(A \wedge B) = p(A|B)p(B) = p(B|A)p(A)$$

- **Conditional independence:** Two random variables X and Y are said to be statistically independent (and is denoted by $X \perp\!\!\!\perp Y$) if and only if $p(X = x \wedge Y = y) = p(X = x)p(Y = y)$ for every value of x and y. Also, X is conditionally independent of Y given Z, and is denoted by $X \perp\!\!\!\perp Y \,|\, Z$, if

$$p(X = x, Y = y \mid Z = z) = p(X = x \mid Z = z)\, p(Y = y \mid Z = z)$$

  for every value of x, y, z. Graphical models, described below, are useful to describe conditional independence of a set of variables. These models can take profit of *gates* [Minka and Winn 2008] to describe a weaker notion of independence known as **context specific independence** which means that some variables are independent for certain values of other variables.
- **Theorem of total probability:** if events $A_1, \ldots, A_n$ are mutually exclusive and $\sum_{i=1}^{n} p(A_i) = 1$

$$p(B) = \sum_{i=1}^{n} p(B|A_i)p(A_i)$$

10 We will avoid distinguishing discrete and continuous random variables and other measure theoretic issues to simplify notation using $p(\cdot)$ for both discrete events and continuous densities, they can be disambiguated by the context. In a similar way, the use of summations and integrals can be used interchangeably.

- **Chain rule:** any joint distribution $p(X_1, \ldots, X_n)$ can be factorized as follows

$$p(X_1, \ldots, X_n) = p(X_1|X_2, \ldots, X_n) \cdot P(X_2|X_3, \ldots, X_n) \cdots P(X_n)$$

Bayes theorem

Bayes theorem

Bayes theorem relates conditional probabilities which are the reverse of each other, it is a consequence of product rule:

$$p(A \mid B) = \frac{p(B \mid A)\, p(A)}{p(B)}$$

For instance, consider a classification problem where there are K classes $C_1, \ldots, C_k$. We would like to know $p(C_k|x)$ but we have estimated the class conditional densities $p(x|C_k)$ instead. Let us also suppose that prior probabilities $p(C_k)$ are also at our disposal. Using Bayes theorem:

$$p(C_k \mid X = x) = \frac{p(X = x \mid C_k)\, p(C_k)}{p(X = x)}$$

and, using the sum rule to express $p(X = x)$ in terms of $p(X = x \mid C_k)$:

$$p(C_k \mid X = x) = \frac{p(X = x \mid C_k)\, p(C_k)}{\sum_k p(X = x \mid C_k)}$$

Bayesian updating

Bayes theorem is used, in the Bayesian interpretation of probability, to "update" our uncertainty given some new evidence. That is, instead of assuming that the parameters θ of our model are *fixed* (but unknown) and that we have to discover them, these parameters are rather considered as random and thus they also have a distribution just like the observed data x:

$$p(\theta \mid X = x) = \frac{p(X = x \mid \theta)\, p(\theta)}{p(X = x)}$$

where:

- $\theta \in \Theta$ are the parameters of the model, they are usually partitioned into parameters of interest and "nuisance" parameters;
- x is the observation;
- $p(\theta)$ is the **prior** distribution of model parameters. It reflects our knowledge about θ being the correct parameters *before having observed the data* and can be used as a regularization term since it is able to model the fact that more complex parameters are inherently less probable (Occam's razor). Quite interestingly, it also allows us to take into account estimations from previous training data (i.e. there can be more or less informative priors);

regularization

Occam's razor

- $p(X = x \mid \theta)$ is called the **likelihood** of θ, and the corresponding function $\theta \mapsto p(X = x \mid \theta)$ is known as likelihood function. This function contains the available information provided by the sample and measures how probable the observation is for different settings of θ;

- $p(X = x)$ is called the prior predictive distribution and can be obtained by marginalization as follows:

$$p(X = x) = \int p(X = x \mid \theta) p(\theta) d\theta$$

- $p(\theta \mid X = x)$ is the **posterior** distribution which expresses the updated epistemological uncertainty about parameters θ after taking the prior and the observation into account.

Using this nomenclature, we can express it as follows:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

or, when the denominator can be considered a normalization constant:

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

Generative vs discriminative models

In the probabilistic setting, generative models, as the name implies, are those where it is theoretically possible to draw samples.[11] When dealing with classification problems, it would suffice to model the conditional probability $p(C|X)$ of the class C given the input data X making it unnecessary to model the distribution of the observed data $p(X)$ for the classification tasks. Models representing just $p(C|X)$ are called discriminative and they only need to represent the distinctive features of each class, disregarding their commonalities.

It is clear that $p(C|X)$ could be obtained from the joint distribution $p(C, X)$ by means of marginalization. Not surprisingly, density estimation is the most general type of learning problem since others can be deduced from it. However, some machine learning paradigms try to avoid to model densities when the final goal is to take a decision.

We have also seen that the other conditional $p(X|C)$, the likelihood which quantifies how likely the observation X is given the class, can be used together with the prior $p(C)$ to perform the classification by using Bayes theorem. This last procedure is also associated to generative models and is sometimes known as *generative factorization*. Both generative and discriminative approaches have theirs pros and cons and, in general, both have their place. Our work is focused on generative models and, among their advantages, they can use different types of techniques and can use different knowledge sources to model and to train, respectively, the different parts. The discussion about the training of generative and discriminative models is delayed to the following section.

generative factorization

11 Let us observe that some models that we can consider generative, as is the case of hybrid HMM/ANN models described in Section 7.5.3, are basically generative models where one component, in this case the model responsible of frame emissions, produces an scaled estimate of $p(\text{frame}|\text{class})$ from $p(\text{class}|\text{frame})$ (which is estimated by a discriminative model) by means of Bayes theorem.

Learning methods

Given an observation x, if we decide to take the action (or class) a, the loss incurred when the true state (or class,[12] since we are classifying) is s is $\lambda(a|s)$. We can estimate the *expected loss*, also known as conditional risk, by considering the probability of all states of nature, given the evidence, as follows:

$$R(a|x) = \int_{\mathcal{S}} \lambda(a|s)p(s \mid x)ds \tag{4.1}$$

Given a decision rule $\alpha(\cdot)$, the average loss[13] on new test data or the *overall risk* is defined as:

$$R = \int_{\mathcal{X}} R(\alpha(x)|x)p(x)dx \tag{4.2}$$

Although there are several criteria to choose a decision rule[14] it is usual to try to minimize the expected loss. The decision rule which selects the class which minimizes this value is known as *Bayes optimal decision rule*:

Bayes optimal decision rule

$$\alpha(x) = \underset{a \in \mathcal{A}}{\operatorname{argmin}}\, R(a|x) \tag{4.3}$$

This decision rule has some specific terms for some particular loss functions. For instance, it is known as *minimum error rate* for the zero-one loss function which tries to predict:

$$\alpha(x) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, p(a|x) \tag{4.4}$$

which, in a generative factorization, and by applying Bayes theorem, can be written as follows:

$$\alpha(x) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \frac{p(x|a)p(a)}{p(x)} = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, p(x|a)p(a) \tag{4.5}$$

Learning these decision rules leads to a density estimation problem. The goal of a learning method consist on choosing a decision rule from an hypothesis space[15] $\mathcal{H}$. This usually corresponds to choose a parameter $\theta \in \Theta$.

The Bayesian interpretation is based on considering θ as another random variable so that Bayes theorem can be used to update our prior belief $p(\theta)$ given some new evidences in the form of the training data $\mathcal{D}$ to obtain an informative prior $p(\theta|\mathcal{D})$. The full posterior distribution $p(\theta|\mathcal{D})$ is used to estimate

$$p(x|a) = \int_{\theta \in \Theta} p(x|a, \theta)p(\theta|\mathcal{D})$$

12 It is convenient to assume, in this case, that actions are isomorphic to states of nature ($\mathcal{S} \equiv \mathcal{A}$).

13 It can be interpreted, in a frequentist interpretation of probabilities, as reducing the average loss over a series of classifications.

14 For instance, *minimax* consists on choosing the label which minimizes the worst case (maximum) loss value.

15 Which do not necessarily contain the true model, since this can be unfeasible in most cases or even counterproductive because a too complex model usually requires a larger training set.

The Bayesian approach, which takes into account the predictions of all hypotheses weighted by their posterior probabilities, is elegant but usually unfeasible.[16] An approximation to the Bayesian approach which produces a point estimate consists in taking the parameter that maximizes the posterior probability $p(\theta|\mathcal{D})$, which is known as Maximum A Posteriori (MAP) estimation. The approach tackled by the frequentist interpretation of probability assumes that model parameters ($\theta \in \Theta$) are fixed albeit unknown. They try to find point estimates by principle and usually define a loss function defined over $\mathcal{D}$ and try to minimize it. For example, a common loss function is the joint likelihood of the training data associated to the *Maximum Likelihood* inductive principle, which consists on choosing the parameters that produce a distribution that gives the observed data the greatest probability or, since the logarithm function is a monotone transformation, maximizes the log-likelihood. MAP generalizes MLE since it coincides with MLE when the prior is uniform.[17] The advantage of MAP estimation, in the general case, it their capability to pull more or less the estimate towards the prior depending on the strength of this prior.

MAP estimation

MLE estimation

Another point of view, in the frequentist setting, consists in observing that the risk functional is based on an unknown distribution p, so that we can replace this risk by the *empirical risk*, which measures the average loss we would have incurred on the data set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, as follows:

empirical risk

$$R_{emp} = \frac{1}{N} \sum_{i=1}^{N} \lambda(\alpha(x_i)|y_i)$$

Choosing the decision rule which minimizes the empirical risk is known as the *empirical risk minimization* (ERM) inductive principle. ERM seems not limited to decision rules based on estimating densities, but a particular case of ERM is MLE when the loss function is the negative log-likelihood.

empirical risk minimization

Since training data is just a set of outcomes,[18] it is important to know how the learning algorithm is influenced by a particular choice of training patterns. If a learning algorithm systematically produces incorrect models for a given observation x no matter the training set, we say that this learning algorithm is *biased* for this particular input. On the other side, if the system trained with different datasets predicts different actions when observing the same observation, we say that the learning algorithm has a high *variance* for this observation. Usually, there exists a trade-off between bias and variance: a learning algorithm with low bias has to fit the data but, if it is too flexible, it will fit too much each particular training set and may even model coincidental or random features (over-fitting) and, consequently, has a high variance.

bias and variance

16 Some particular cases, for instance when using conjugate priors, can be solved analytically. In other cases it is possible to make use of sampling (e.g. Monte Carlo) methods. Let us remark that the benefit of the Bayesian approach might be obtained with the alternative, more efficient and perhaps simpler use of ensemble techniques.

17 Moreover, training with MLE with regularization can be similar, in some cases, to MAP estimation when the regularization term can be viewed as a prior.

18 We are assuming, in this introduction, an observational setting where different samples are obtained in a i.i.d. way, but, as mentioned before, there exist other possibilities such as active learning paradigms where the learning system proposes what it consider to be the most informative samples.

Relating ERM, it is also usual to include a regularization term to improve generalization. This term penalizes the use of more complex parameters and this approach is known as regularized risk minimization.[19] The weight of the penalty term in regularized risk minimization can be used to control the bias-variance trade-off.

To sum up, we have two ways for constructing the decision rule: we can try to map the input features onto a class label without resorting to the underlying distribution or we can base it on an estimation of a probabilistic model. We will focus our work on the second approach where it is possible to distinguish between generative and discriminative models as discussed before. Since this discussion preceded the short review of learning methods, we can now remark that although generative models can be trained using loss functions based on the joint likelihood of the observed data, it is also possible to train them using a discriminative training objective function which tries to increase the posterior probability. That is why we can distinguish between generative and discriminative training of generative models. Generative models trained discriminatively are not meant to be descriptive (used to draw samples from them), but they can perform often as well as discriminative models, which makes them, in some way, the best of two worlds although both generative training and the use of inherently discriminative models remain completely worthwhile.

generative vs discriminative training

Both generative and discriminative probabilistic models try to represent a (usually multivariate and quite complex) joint distribution or a conditional distribution. Models based on a generative factorizations also require the estimation of likelihoods. There exists a huge quantity of parametric and non-parametric methods and models to these ends but, in practice, we have to limit our scope to what can be learned.[20] Besides tying parameters, a reasonable procedure to construct tractable and learnable models is by means of probabilistic graphical models described below. They are intended to factorize complex models and there exist several inference and learning algorithms but, as we will see, their graphical structure have to be supplied with information about the specific components. Before deepening into graphical models, we have to remark that these components usually refer to some well known probability distributions and models which are extensively studied in the general bibliography provided at the beginning of this chapter [Bishop 2006; Duda *et al.* 2001; Murphy 2012]. In particular, the models most commonly used in our field[21] mixtures of parametric distributions and artificial neural networks (ANNs). The first ones are distributions taken from a parametric family, specially the exponential family and, notably, the Gaussian distribution which leads to the largely used Gaussian mixture models (GMMs). GMMs can be trained using the EM algorithm and model likelihoods. On the one side, ANNs are usually trained by means of gradient descent and are normally used to estimate posteriors.

density estimation

19 The review of Structural risk minimization [Vapnik and Vapnik 1998] based on bounds on the distance between the expected risk and the empirical risk and other concepts such as the capacity of the hypothesis space is out of the scope of this little review.

20 Which remembers the following quote from [Domingos 2012]: *the key question is not "Can it be represented?", to which the answer is often trivial, but "Can it be learned?".*

21 They are related with the emission of frames in segment models of Section 7.5 where other model types will also be enumerated.

4.1.1 (Probabilistic) graphical models

Working with joint distributions defined over many random variables is intractable. A possible solution when there are conditional independence assumptions between variables is the use of probabilistic graphical models. They are a particular case of the broader concept of graphical model (GM). Graphical models other than probabilistic ones include constraint networks, cost networks and influence diagrams. Even in the probabilistic case, they can be augmented with decisions and rewards, as is the case of partially observable Markov decision processes and influence decision diagrams.

GM is a broad concept

The vast literature about graphical models proposes a large number of algorithms which, in many cases, subsume a much larger number of previously well known algorithms [Aji and McEliece 2000; Kschischang *et al.* 2001] hence offering an interesting and unifying point of view which covers many apparently unrelated algorithms of different fields. Many of these algorithms can be described as message-passing algorithms [MacKay 2003; Chapter 16]. It should be remarked that, nevertheless, this does not invalidate other alternative viewpoints. There exist exact and approximate algorithms and many of them basically try to take profit of factorizations and try to avoid the repeated computations of intermediate sums which are reused many times. Before discussing inference, let us first concentrate on representation.

In general, graphical models allow us to represent a global function defined over a finite set of variables as a factorization (e.g. as a product) of local functions defined over (ideally much smaller) subsets of the global function domain. An assignment of variables is called configuration. A *commutative* combination operator is used as the product to construct the global function. Let us observe that this operator is not necessarily the product nor has to be defined over real numbers (it could be the *and* operator defined over Boolean values, for instance), but the presence of a zero value allows us to distinguish *valid* configurations which are those leading the global function to non-zero values.

in a nutshell, it's all about factorization

A general way to describe these factorizations, not only for the probabilistic setting, are factor graphs [Kschischang *et al.* 2001] described below. In the case of probabilistic GMs the global function is a joint distribution, although there also exists GMs describing conditional distributions. The graph topology of probabilistic GMs defines a set of CI assumptions and implicitly represents the family of joint distributions consistent with them. These graphs must be accompanied with specific definitions of factors to represent specific distributions. A graphical model with less CI assumptions or with a coarse factorization may also represent a given distribution at the expense of a loss of efficiency in representation, inference or reasoning capabilities. If the factorization point of view is appropriate to reason about inference algorithms, thinking about CI assumptions seems better suited to design models.

define joint distribution families

These GMs, defined over a finite set of random variables, are what we will term "ground" graphical models. There also exists mechanisms, (delayed to a next subsection) to extend these models to time series, sequences and, in general, to represent more or less implicitly a family of ground models. The following list describes the most common GM types:

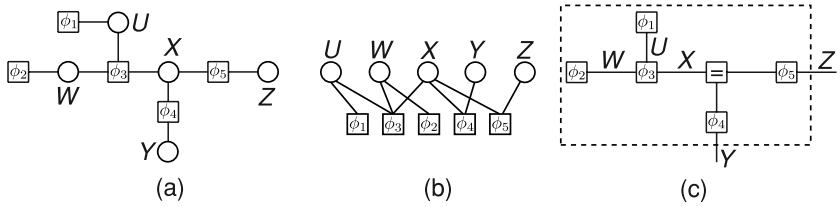


Figure 33: Factorization $p(U, W, X, Y, Z) = p(U)p(W)p(X|U, W)p(Y|X)p(Z|X) = \phi_1(U)\phi_2(W)\phi_3(X, U, W)\phi_4(Y, X)\phi_5(Z, X)$ represented as factor graphs using different notations: (a) The classical representation of factor graphs as bipartite graphs (factor nodes represented as squares and variable nodes represented by means of circles), (b) the same factor graph depicted in a different arrangement to emphasize the fact that they are bipartite graphs, and (c) the factor graph in *normal* or Forney style. As can be observed, an equality constraint node $\boxed{=}$ is used to replicate the variable X. We can also observe the presence of half-edges (Y and Z) which define an external function[23] $\mathcal{G}(Y, Z)$ so that the entire factor graph can be considered as a function, well suited for hierarchical representations (Figure adapted from [Loeliger 2004]).

explicit factorization

FACTOR GRAPHS (FGs a.k.a. valuation networks[22] [Shenoy 1992]) are undirected graphs which explicitly represent the terms of the factorization. These factors or local functions are nodes of the graph. There exists at least two different factor graph notations. In the original one, another type of node corresponds to function variables. Since function nodes are only connected to variable nodes and vice-versa, factor graphs are bipartite, as illustrated in Figures 33 *(a)* and *(b)*. Each function node ϕ_k depends on the subset of variables X_{C_k} which it is connected to in the graph. The joint distribution of the factor graph is defined as the product:

$$p(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{k=1}^{K} \phi_k(X_{c_k})$$

partition function

In the case of probabilistic factor graphs, the constant $1/Z$ is introduced to assure the function is normalized. The value Z is called the *partition function* and is defined as follows:

$$Z = \sum_{X_1, \ldots, X_n} \prod_{k=1}^{K} \phi_k(X_{C_k})$$

This constant is guaranteed to be 1 in some cases, for instance when factors are locally normalized. In other cases, the computation of the partition function corresponds to one of the main inference problems, namely the computation of marginals.

A FG is essentially a graph representation of an hypergraph where it is possible to consider that either variables or factors are the nodes and their counterparts are hyperedges. The *primal graph* of a FG is an undirected graph where vertices represent variables and edges connect two variables if they appear in the

22 This nomenclature is most known in the broader area of valuation algebras.

same scope of some factor. Similarly, the *dual graph* is an undirected graph where vertices represent factors and edges connect factors that share a variable in their domains.

conditional independence

Relating conditional independence, two variables are neighbors if they share a common factor (they are neighbors in the primal graph). A path of variables is a sequence of neighboring variables (a path on the primal graph). Two variables X and Y are conditionally independent given a set of variables $\mathcal{V}$ (denoted $X \perp\!\!\!\perp Y \mid \mathcal{V}$) if every path between X and Y contains a variable $V \in \mathcal{V}$ (global Markov property). In particular, a variable is conditional independent of others given its neighbors (local Markov property).

energy functions

Under quite general conditions when it is posible to restrict local functions to be strictly positive, it is possible to define them by means of energy functions $\phi_k(X_{C_k}) = \exp(-E_k(X_{C_k}))$. Energies can be interpreted as negative log-likelihoods and the energy of the global function is the sum of the energies of the local functions. Energy based models [LeCun *et al.* 2006] can be used in both probabilistic and non-probabilistic approaches (see Section 4.4.4). An important family of energy functions have the form of a linear combination of feature functions $E_k(X_{C_k}) = \sum_j \theta_j f_j(X_{C_k})$.

An alternative notation to factor graphs is known as *normal* (factor) graphs or also as *Forney style* factor graphs [Forney 2001]. In this notation, variables are not explicitly represented as nodes, they are rather represented as edges, as illustrated in Figure 33 *(c)*. Besides edges connecting two nodes,[24] there also exists half-edges (a.k.a. dangling edges) which are only connected to one node. Normal factor graphs define a function called *exterior function* (a.k.a. sum-of-products form) which is the product of its local functions together with the marginalization of the internal variables (variables not associated to half-edges). In this way, they represent a function defined over the half-edgtes which is well suited for hierarchical representations (sub-graphs represented by means of boxes) as in block diagrams.

exterior function

hierarchical representation: modularity

The main advantages of factor graphs, compared with other representations, are the easy of description of the associated inference algorithms and their expressive power[25] compared with Markov Random Fields and Bayesian Networks (defined below) as depicted in Figure 34. Factor graphs allow us to represent arbitrary factorizations of the global function and, indeed, there exist factor graphs which cannot be represented by means of either BNs or MRFs. Directed factor graphs [Frey 2002] constitute a valuable and interesting extension which allows the specification of causal and non-causal knowledge in the same model.[26]

24 The limitation of normal graphs that no variable can appear in more than two factors can be circumvented by means of auxiliary variables and the inclusion of equality constraint nodes, as illustrated in Figure 33 *(c)*.

25 By expressive power we mean their capability to represent both a particular factorization of the joint distribution and a given set of conditional independence assumptions.

26 It is possible to convert MRFs and BNs, described below, to directed FGs without loss of semantics. An unrelated proposal is [Dietz 2010].

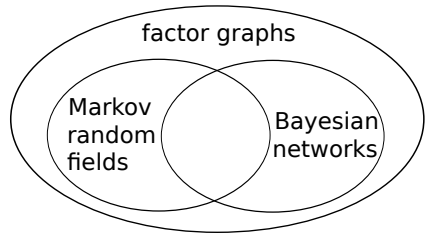


Figure 34: Relationship between Bayesian Networks, Markov Random Fields and factor graphs relating their respective capabilities to represent both the set of conditional independencies and the same factorization of the global function or joint distribution (figure inspired by [Frey 2002; Fig. 1])

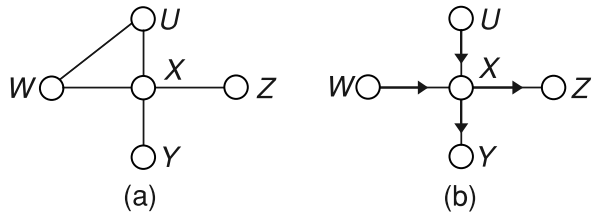


Figure 35: Factorization $p(U, W, X, Y, Z) = p(U)p(W)p(X|U, W)p(Y|X)p(Z|X)$ of Figure 33 represented by means of: (a) a Markov Random Field, and (b) a Bayesian Network (Figure adapted from [Loeliger 2004]).

MARKOV RANDOM FIELDS (MRFs) are defined as undirected graphical models where nodes represent random variables and the undirected links represent a (possibly non-causal) relationship between those variables. Links are interpreted in such a way that CI of variables in a MRF follow the Markov property. This property is similar as described in FGs taking into account that, now, the concept of neighborhood is that of the graph. A variable in a MRF is CI of the others of the graph given the set of its neighboring variables (this set is known as the Markov blanket of the node). In both cases, the basic idea is that neighbors of a node contain enough information to predict its value.

A common way to factorize the joint distribution consists in associating a local function to each maximal fully-connected subgraph of the graph (the set of maximal cliques), as depicted in Figures 36 *(a)* and *(b)*. As can be observed in Figure 36 *(c)*, there exist factorizations which can be expressed by means of FGs and which only admit a coarser representation when using MRFs.

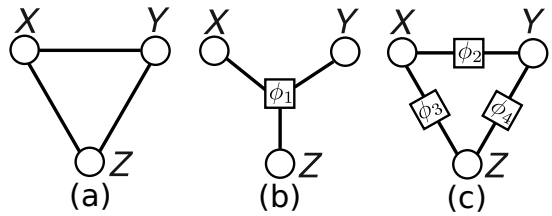


Figure 36: Ambiguous factorization of MRFs: (a) a MRF, (b) a factor graph whose factorization is compatible with the MRF of the left. This factorization has only one factor $\phi(X, Y, Z)$ associated to the maximal clique, and (c) another factor graph which would also correspond with the MRF of (a), in this case with three factors $\phi_2(X, Y)$,$\phi_2(X, Z)$ and $\phi_4(Y, Z)$.

BAYESIAN NETWORKS (a.k.a. belief networks) are acyclic directed models where the factorization of the joint distribution is expressed as a product of conditional distributions. These factorization can be always obtained by applying the chain rule decomposition[27] to the joint distribution, which is not unique since it depends on the variable order. As with MRFs, nodes represent random variables but, in this case, directed edges $X \to Y$ represent the presence of X in $p(Y|\cdots)$. The joint distribution can then be expressed as:

$$p(X_1, \ldots, X_n) = \prod_{k=1}^{K} p(X_i|\text{parents}(X_i))$$

The CI assumptions, as with other GM types, is defined by the absence of edges. The local semantics states that a node is CI from its non-descendents given its parents. The equivalent of a minimal Markov blanket for a node is given by the union of its parents, its children, and the parents of its children (a.k.a. co-parents). A variable is independent of the rest of the variables given its Markov blanket. In general, to reason about independence between two variables given a disjoint variable subset $X \perp\!\!\!\perp Y \mid \mathcal{V}$ is similar to the previously described GM types in the sense that it is based on the properties of paths between X and Y although, in this case, it is more cumbersome. The concept is known as *d-separation* and states that $X \perp\!\!\!\perp Y \mid \mathcal{V}$ holds if every undirected path between X and Y satisfies that:

d-separation

1. whenever the following heat-to-head configuration appears $X \to Z \leftarrow Y$, then Z or one of its descendants belongs to $\mathcal{V}$;
2. in any other case, the nodes of the path are not in $\mathcal{V}$.

To interpret a BN as a FG is suffices to consider $p(X_i|\text{parents}(X_i))$ as factors. It is possible to obtain a MRF capable of expressing the same joint distribution by removing the direction of edges and by including additional edges connecting all pairs of parents of each variable in order to assure that all the variables of each conditional distribution belong to the same maximal clique and, hence, is contained in the same factor of the MRF.

CONDITIONAL RANDOM FIELDS (CRFs) are a type of discriminative and undirected probabilistic GM used to describe a conditional distribution $p(Y|X)$ over the set of variables $V = X \cup Y$ where X is the set of input variables, which are observed, and Y is the set of output variables. The original description [Lafferty *et al.* 2001] is similar to MRFs to the extent that the graph only represents variables, not factors.[28] A more recent tutorial describes CRFs just as a particular case of factor graphs: a factor graph G is a CRF if its factorization can represent $p(Y|X)$ [Sutton and McCallum 2006].

27 So that BNs are acyclic by construction and the normalization term and the associated partition function are not required.

28 The output variables obey the Markov property conditioned on observed ones.

Probabilistic inference

So far we have described graphical models from a representation point of view paying special attention on their capabilities to represent a global function in terms of factors and particularly, for probabilistic graphical models, to represent either a joint or a conditional distribution satisfying some independence properties. Besides the representation problem, we should also deal with learning and with inference. Learning comprises determining the best suited graphical representation (identification of variables and the implicit set of independence assumptions and possible factorizations, the general form of these factors) and the estimation of factor parameters from training data. On the other side, inference refers to reasoning about the probabilities of one subset of random variables given values of another subset.

nomenclature

Relating the nomenclature usually found in the literature for inference tasks, the terms *explanation set* and *query variables* are used to denote the set of variables related to the query, normally a subset of hidden variables. Depending on the query type, we usually seek to know either their posterior values or the most probable instantiations given the value of other variables. The *evidence* denotes the variables whose value is known. Given a query, it is posible to determine the set of variables relevant for this query, this subset is known as *requisite variables* and does not include necessarily all evidence variables. The rest of variables (other than the requisite ones) are usually known as nuissance or irrelevant variables. Requisite variables whose value is not specified nor required can be marginalized.

There exist some canonical or coined problem types:

- to compute posterior marginals, i.e. to determine the posterior probability distribution $p(C|E)$ of a set of variables C given evidence E for other subset of variables. This problem type can be further divided into the following ones [Kwisthout 2009]:
  - *exact inference*[29] tries to determine the value of $p(C|E)$;
  - *positive inference* tries to determine when $p(c|e) > 0$; and
  - *inference* is related to the general query $p(c|e) > q$.

  Some particular cases of exact inference are:
  - *normalization problem*: to compute the partition function Z;
  - *marginalization problems*: to compute the marginals for each individual variable.
- to find the most probable explanation (MPE) where, given some evidence, we seek to find the maximum probability assignment to the remaining variables;
- to find the maximum a-posteriory hypothesis (MAP) is similar to MPE where we are only interested in the assignment for a subset of query variables. Although this problem, usually known as *partial MAP*, can be considered as a generalization of MPE, it is convenient to distinguish MPE as an independent problem type since there are more efficient algorithms to solve it.

29 There is a clear nomenclature clash since this term is also used to refer to inference algorithms which are exact, in opposition to approximate ones.

For graphical models augmented with decisions and rewards, it is interesting to find out which decisions would lead to the maximum expected utility (MEU). In some cases, the observation process to obtain new evidences has a cost so that it has to be determined which evidence to seek next.

It is worth noting that posteriors and MAP hypothesis can be described in a quite abstract way in terms of semiring[30] operations so that both problem types correspond to the same computations performed on two different semirings using either the sum or the maximization semiring summary operator. In this way, most inference problems can be described in terms of *marginalizing a product function* (MPF).

There exists a large number of both exact and approximate inference algorithms in the literature for probabilistic graphical models which basically try to solve the MPF problem in a more efficient way than a simple or naive brute force computation by exploiting the distributive properties of semiring operations and the fact that the graphical model represents a product of factors. For instance, in order to sum out or to marginalize a variable X from a product of factors we do not need to perform the sum operation on the entire product but only to take into account those factors that contain X in their domain.

The complexity of most exact algorithms depends only on the graphical model structure (structure-based algorithms, whose cost is mainly related to the graphical model *treewidth* discussed below), in other cases the cost also depends on the special features of some factors (known as *parametric* or *local structure* [Darwiche 2009; Chapter 13]).

This short review is focused on the algorithmic versions tailored for FGs. We focus on FGs for two different reasons: many other graphical models types (not only probabilistic ones) can be converted into FGs and these algorithmic versions seem, in our humble opinion, more elegant and simpler to describe and to understand. Another goal of this brief review is to try to describe the relationship between the summary-product (or belief propagation) algorithm and the junction tree algorithm, a relationship which is not usually found in the literature, a notable exception being [Darwiche 2009].

It is interesting to take also into account, for computational purposes, if we are trying to solve just a single query or several ones. There exist *query sensitive* inference algorithms which have to be executed again and again for each different query. On the other side, there are inference algorithms which allows the simultaneous execution of several queries. It may be more efficient, for instance, to jointly compute several marginals since some computations can be reused and the overall cost is reduced. Indeed, some works such as [Kask *et al.* 2005] formalize the inference problem in terms of a set of sets of query variables. This formalization also states the choice of a marginalization operation related with the choice of a semiring summary operation described before.

Most exact inference algorithms are based on the concepts of elimination (either factors or variables) and conditioning (reasoning by

30 Let us also observe that there a framework even broader than that based on semirings: *valuation algebras* [Pouly and Kohlas 2012] which have associated quite general exact inference algorithms where some widely known probabilistic inference algorithms are particular cases.

cases by assuming that some variables are assigned some values). Applied to FGs, there exist some transformations which can be performed without affecting the global function semantics [Kschischang *et al.* 2001; Section VI] such as clustering several factors (replacing these factors by a new one which is connected to the set of variables of the scopes of the replaced factors). Another interesting fact is that we can trivially marginalize (by means of marginalization or summation) those variables appearing only in one factor without affecting the rest of the graph. The combination of these two techniques can be used to remove variables or factors from a FG in several ways. In particular, we can highlight:

VARIABLE ELIMINATION it is possible to eliminate a variable by first identifying the set of factors that contain this variable in their domain. Once these factors are clustered, the variable can be removed by marginalization. This process is applied over all requisite variables not appearing in the query, on at a time;

FACTOR ELIMINATION once a factor is chosen to be eliminated, it is desirable to remove (by means of marginalization) all variables which only appear in the scope of this factor. The resulting factor is clustered to another other factor of the graph in order to reduce the number of factors. This procedure, as with variable elimination, is applied iteratively until just one factor remains which contain in its scope a superset of the query variables. The result is finally obtained by removing the non-desired variables.

These approaches, described in this way, are query sensitive since we have to execute the algorithm again and again for each different query. This limitation will be overcome below.

Let us observe that several factors are removed in a single step of variable elimination whereas, in general, several variables may be removed with factor elimination. The cost of each operation depends on several considerations such as the representation of factors, the number of variables on the scopes of factors involved in the operation, the number of variables to be marginalized and by the size of their domains. The cost is usually specified for factors described explicitly with tables defined over variables of a finite domain. In this case, the temporal cost is $O(k^w)$ being k the size of the domain of the variables and w the total number of variables. The spatial cost depends on the number of variables which are not marginalized. It is possible, in other cases, to represent factors analytically. For instance, factors representing functions from the exponential family benefit from the fact that combination and marginalization preserve the membership to this family and marginalization might be performed by means of analytic integration.[31]

Although the variable elimination ordering does not affect the correction of the algorithm, different orders may lead to different costs since the number of variables of intermediate factors may vary. The size of the largest (intermediate) factor determines the cost and is known as the *width* of the particular elimination order. This value

31 This is related with the message types of message passing algorithms discussed below, but see[Loeliger 2004; Section 4.4].

can be obtained from the primal graph of the FG. Each time a variable is removed from the FG, the primal graph can be updated by connecting non-adjacent neighbors of the corresponding vertex variable and then removing this vertex. The edges added in this way as known as fill-in edges. The width of the elimination order depends on the degree of the maximum number of neighbors, in the primal graph, of the eliminated variables. The minimal width of a graph, taking into account all possible elimination orders, is known as the *treewidth* and is a lower bound on the complexity of the variable elimination inference algorithm.[32] Unfortunately, determining the best ordering (where the width reaches the treewidth), in the general case, is NP-complete [Arnborg 1985], although there are some heuristics for obtaining good elimination orders.

Relating the implementation of variable elimination algorithms, there is a technique to easily identify the set of factors associated with the variable to be eliminated: the bucket elimination algorithm [Dechter 1999] distributes the factors of the FG into sets (or *buckets*) associated each one to a FG variable in such a way that each factor is placed in the bucket of the variable of its scope which is eliminated in the first place. The factors to be taken into account when removing a variable coincides which those found in its bucket. The resulting factor obtained by combining them and marginalizing the removed variable is placed, again, in the bucket associated to the variable from its domain which is eliminated in the first place (a variable not processed yet). This process preserves the property that a variable to be eliminated contains the required factors in their bucket. Another variants of variable elimination are the fusion elimination [Shenoy 1992] and the collect algorithm [Pouly and Kohlas 2012] which is based on join trees discussed below.

bucket elimination

For the particular case of NFGs, [Al-Bashabsheh 2014; Chapter 5] shows that any query can be converted into the evaluation of the exterior function of a modified NFG. In this way, the evaluation of the exterior function may subsume other inference problems. It is then possible to compute the exterior function by removing all edges in order to obtain a single factor with just half-edges (dangling edges). The proposed technique consists of merging adjacent factor in an iterative way until just one factor remains which seems a variant of factor elimination. In order to reduce two adjacent factors, we have to marginalize all variables connecting them. As with variable elimination, the cost depends on the order the factors are merged. When applied at the block level (merging a factor with all its neighbors) over the equality constraint nodes (used to replicate variables) this is essentially the previously described variable elimination on FGs.

If a variable ordering is a sensible way to specify how to perform variable elimination, in the case of factor elimination it is reasonable to specify the elimination strategy by means of a tree (called *elimination tree*) where each node contain a factor[33] to be eliminated which is ac-

32 This parameter also appears in other representations suitable for probabilistic inference described below (e.g. factor elimination trees, jointrees) and has connections with many other algorithms not necessarily related with inference.

33 Several factors can be attached to the same tree node, although in this case this would be equivalent to clustering all these factors in the FG. Also, a node with no factor on it

cumulated on the tree node parent [Darwiche 2009; Chapter 7]. When taking a tree node as root, factors appearing as tree leaves can be removed and clustered to their parents on the tree until only the root remains. It is possible to determine efficiently which are the variables to be marginalized in each step and which are the variables of the scope of a factor (when it is ready to be eliminated) by means of two notions defined over factor elimination trees: *separators* and *clusters*, respectively.

SEPARATORS are associated to factor elimination tree edges. Since the edge divides the tree into two parts, the separator is the set of variables which appear in both sides in some scope of some factor, so we should marginalize all variables not appearing in the separator;

CLUSTERS are associated to tree nodes. Each cluster is computed by joining the set of variables of the factor of the node together with the separators of its adjacent edges.

The factor elimination algorithm can be formulated as a message passing algorithm over the factor elimination tree where tree nodes can be seen as processors which send messages between them over the tree edges. These messages are essentially the factors themselves that are accumulated in the receiver. The rules to implement the message passing algorithm are quite simple on a rooted elimination tree:

- each node accumulates (multiplies) the messages received from their children into its own factor;
- when all messages have been received, the accumulated factor is projected[34] to the set of variables of the separator between the current factor and its parent and the result is sent to the parent.

This algorithm starts at the tree leaves and the root is chosen so that the query variables are contained in the root cluster. The computation of all the marginals associated to each tree node cluster can be computed in a very efficient way by caching intermediate results sent over the separators and by slightly modifying the message passing rules so that the role of parent is any neighbor and messages are sent when all messages has been received from the rest of the neighbors. The cost of computing all tree node marginals is reduced from $O(n^2)$ messages to $O(n)$, where n is the number of tree nodes (essentially the number of factors). The cost of computing these messages, and hence the temporal cost of the algorithm, depends on the cluster sizes (as described before when discussing the variable elimination algorithm). This cluster size is bounded by the treewidth. The spatial cost may also depend on the cluster size or on the size of the separators, depending on the particular message passing architecture [Kask *et al.* 2005].

This type of algorithm has been traditionally explained in terms of a jointree, also known as junction tree or tree-decomposition. Although

would be equivalent to an trivial function assigning the neutral element of the product semiring operator.

34 Projection over a set of variables means marginalizing all variables out of this set.

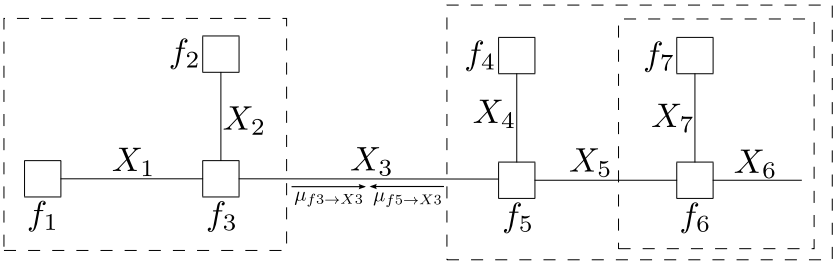


Figure 37: Cutting an edge in a cycle-free NFG divides the factor graph into two separate subgraphs. In this example, breaking the edge associated to the variable X_3 divides the graph into two parts whose exterior functions are represented as $\mu_{f_3\to x_3}(x_3)$ and $\mu_{f_5\leftarrow x_3}(x_3)$ respectively (figure adapted from [Loeliger 2004]).

we will not enter into details, it suffices to know that algorithms described in terms of factor elimination trees and jointrees are essentially equivalent. In particular, the cluster-tree elimination described in [Kask *et al.* 2005] for tree-decompositions is very similar to the factor tree elimination algorithm for computing all marginals described so far. Moreover, it is possible to obtain, at a reasonable cost, a factor elimination tree, a join tree or a variable elimination ordering from any of the other representations[35] in a process that preserves the width [Darwiche 2009; Chapter 9].

Quite interestingly, the factor elimination formulation of the junction tree algorithm brings an additional insight by establishing a connection with the summary product algorithm for cycle-free factor graphs [Kschischang *et al.* 2001]. This algorithm, also known as belief propagation, has two prominent instances known as sum-product and max-product which vary in the semiring summary operator.

One of the more clear expositions of the summary product algorithm is the version for NFGs. A NFG is cycle-free if each edge divides the graph into two disconnected parts. Since cutting an edge can also be viewed as breaking it into two half-edges, it is possible to consider each part as a NFG whose exterior function (over the just created half-edge and assuming no other half-edge) is the marginalization of the rest of variables in this part of the NFG. In this way, the marginal for each variable is the product of the two exterior functions of the subgraph this variable connects. In the example of Figure 37 we can observe that $p_3(x_3)$ (marginalizing all variables other than x_3 in the global function) can be computed as $p_3(x_3) = \mu_{f_3\to x_3}(x_3)\mu_{f_5\leftarrow x_3}(x_3)$. These functions can be computed recursively[36] so that $\mu_{f_5\leftarrow x_3}(x_3) = \sum_{x_4,x_5} f_5(x_3,x_4,x_5)\mu_{f_6\leftarrow x_5}(x_5)\mu_{f_4\leftarrow x_4}(x_4)$ where $\mu_{f_4\leftarrow x_4} = f_4$. The summary product algorithm is also formulated as a message passing

35 Indeed, [Darwiche 2009; Chapter 9] also discusses an additional data structure for the recursive conditioning algorithm not detailed in this review.

36 This is related with the correspondence between the grouping of parts of a cycle-free NFG into nested non-overlapping boxes and the way of grouping the corresponding factors by means of brackets, what is known as the "closing boxes metaphor". In the example of the figure, the global function is defined as $f(x_1,\ldots,f_7) = f_1(x_1)f_2(x_2)f_3(x_1,x_2,x_3)f_4(x_4)f_5(x_3,x_4,x_5)f_6(x_5,x_6,x_7)f_7(x_7)$. On the other side, the dashed boxes depicted in the figure correspond to the factorization $(f_1(x_1)f_2(x_2)f_3(x_1,x_2,x_3))(f_4(x_4)f_5(x_3,x_4,x_5)(f_6(x_5,x_6,x_7)f_7(x_7)))$.

algorithm where nodes (factors) send messages over edges (variables) and these messages are the μ functions described above. It turns out that the summary-product coincides with the factor elimination algorithm when using the factor graph itself as its elimination tree.

This algorithm seems to be invented/discovered many times and independently of junction tree algorithms, is exact for cycle-free graphical models. Unfortunately, the relationship between both types of algorithms is not usually highlighted in the literature. Viewing summary product as a particular case of factor elimination shows that other marginalizations besides individual variables are possible: clusters associated to nodes.[37] Although summary-product is not exact or may even not make sense[38] for cyclic FGs, it is used in an iterative way leading to the *loopy* version of summary-product. This process may or may not converge to a fixed point and, even if it converges, the result may be incorrect, but in practice it brings surprisingly good results and is considered an approximate technique.[39]

conditioning

The efficiency of the summary-product, compared with the cost of the general case for cyclic FGs, has motivated the use of the conditioning (as stated before, reasoning by cases by assuming that some variables are assigned some values) and other techniques (e.g. joining factors and variables) to break cycles and obtain a cycle-free graph where the summary product can be used. The use of conditioning for this purpose is known as *cut-set conditioning*. This technique has not to be confused with other prominent use of conditioning, known as *recursive conditioning*, which tries to split the FG into disconnected parts which can be solved separately.

HMM, described below, constitute a notable example of cycle-free graphical models and we will show that the computation of the forward, backward and Viterbi probabilities are just a particular case of the summary-product messages, but it is convenient to see first dynamic graphical models since HMMs constitute a particular case.

Beyond ground GMs

Graphical models described so far seem a convenient way for expressing a joint or a conditional distribution over a finite and fixed set of random variables. However, they may be improved and extended in a number of ways by means of some mechanisms, usually accompanied with a graphical notation, to some palliate some of their weakness. The purpose is twofold: On the one side, templates and time slices are used to compactly and implicitly represent a set of *ground*[40] graphical models. On the other side, the purpose of gates is to represent context specific (or sensitive) independence, which is a weaker notion of independence where some variables are independent for certain values of

37 Also, factor elimination may benefit from the observations of summary-product: messages sent between factors, when the edge may break the graph into two separate subgraphs, can be used to marginalize over the variables of the edge separator.

38 Rules should be changed since there may be no leaf nodes where the process may start. They may, for instance, start with a unitary function (neutral element in the product of functions).

39 There exists several works studying this algorithm from a theoretical point of view, some have established a connection with the Bethe free energy, but there does not seem to be a conclusive understanding for the moment.

40 Those we have reviewed so far, a.k.a. propositional as opposed to first-order.

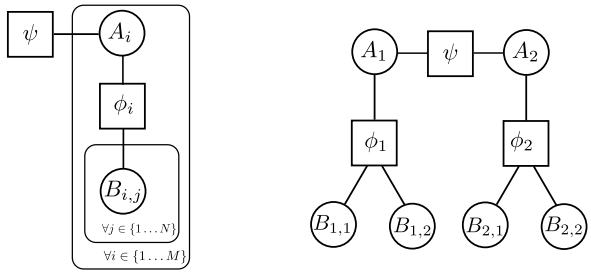


Figure 38: On the left: a factor graph (depicted with classical style, not with Forney style) with two plates one nested inside the other. On the right: an expanded FG equivalent to the left one using values $M = 2$ and $N = 2$.

other variables. Let us briefly overview these two basic mechanisms for replication and conditioning, but remark that there exists other alternatives[41] and that our focus, as with inference, is on factor graphs:[42]

PLATES are a convenient notation to compactly describe repeated structures. A clear example is the replication of a part of the model, which may be convenient to represent mixture models. Plates are graphically depicted as boxes surrounding a part of the GM with an indication in the lower right corner indicating how many copies are created and an index which can be used by the components of the graph inside the plate. This index is required since plates can be nested or can just intersect with other plates, as illustrated in Figure 38. The factors and variables inside several boxes may contain only some of these indices to indicate to which plates they belong to and hence how they are replicated. Connections with a non-replicated variable (e.g outside the plate) can be useful to represent shared parameters;

GATES allows us to represent several configurations which depend on the value of a context variable. As with plates, they are represented by means of a box (in this case, depicted with dashed lines to distinguish them from plates) which surrounds a part of the FG and which contains a label with the value of a key.[43] The box is graphically connected to a selector variable and the meaning of the gate is as follows: factors become the neutral product element when the selector variable is different from the label key but remain unchanged (as without gates) otherwise. For variables the meaning is similar but they are considered to have a default value (zero or false) when the selector variable is different from the label key. [Minka and Winn 2008] explains how different inference algorithms can be adapted for the use of gates. Several gates with different label keys may be associated to the same selector variable, as illustrated in Figure 39. Gates can also be used together with plates.

41 For the case of conditioning, gates seem to have overcome previous alternative proposals, although switching parents are briefly described below when discussing dynamic Bayesian Networks. Relating replication, it is even possible to define ground models imperatively [McCallum *et al.* 2009].

42 Specially for gates, plates are commonly used with BNs as well.

43 A default value is considered for the key when the label is absent.

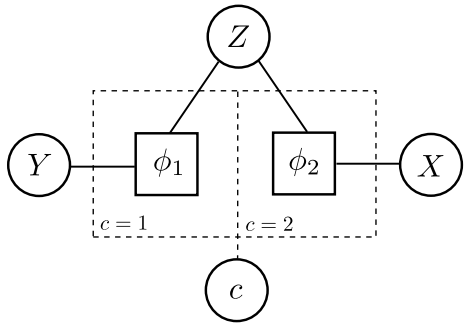


Figure 39: Gate notation. A factor graph with two gates associated to the same variable c. Factor ϕ_1 becomes the unit neutral function when $c \neq 1$, the factor ϕ_2 has a similar treatment for $c \neq 2$.

Dynamic graphical models (DGMs)

Graphical models have been introduced as a useful way to describe statistical models when the number of random variables is known in advance, specially in cases where data is not fully observable. It is therefore highly desirable to be able to adapt these models to variable length sequences, our matter at hand. Although plates constitute a practical mechanism to deal with repeated structures, the use of graphical models for structured data represents a challenge since not only the number of variables may vary (depending on the instance) but they are also related in a complex (albeit regular) way. For the case of sequences, typically focused on time series, the use of plates with a time index would seem a way restricted to parametric trajectories.

An alternative to parametric trajectories is to represent the dynamics of a system evolving in time. Indeed, the term "time slice" is frequently used to denote a template graph representing a snapshot or the state of the system to be modeled in a particular moment.[44] In a 2-time-slice, there exist edges connecting variables from the same time slice (intra-slice edges) and from one time slice to the next one (inter-slice edges). Since this structure is repeated over time, it suffices to describe the topology of time slices by means of a template which is repeated or "unrolled"[45] in order to produce a probabilistic network of the size of observations, as depicted in Figure 40. Besides this "time slice" (also known as "chunk") template, two optional templates known respectively as "prologue" and "epilogue" can also be used at the beginning (to specify the initial conditions) and at the end (to represent the state of the system after the last moment for which there is an observation). This temporal modeling basically assumes that the evolution of the state of the system is Markovian, meaning that it only depends on the previous state.[46] This time-slicing technique has traditionally

44 Or, generally, sequence position where the system does no represent time series.

45 Which reminds us of the unrolling process of backpropagation through time algorithm used to train recurrent neural networks and also the patterns generated by decorative paint rollers and the garlands of paper with people motifs.

46 On a finite set of previous states given by the order of the model, although they can be emulated by lower order Markov models, so that we can assume that it may only depend on the previous state provided it has a representation rich enough.

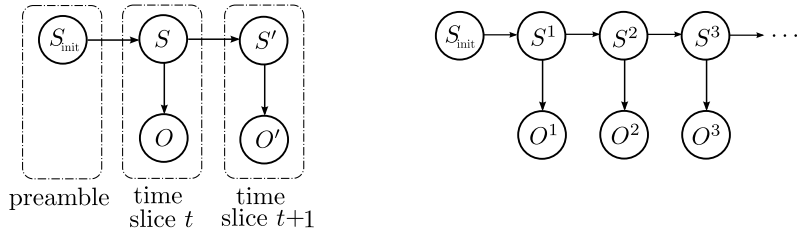


Figure 40: Unrolling HMM represented as a DBN. On the left: the DBN representation of HMMs with an optional preamble and time slice template illustrating intra-slice and inter-slice edges. On the right: the unrolling process leads to a lot of *ground* BNs, each one for a different length.

been applied on Bayesian Networks, leading to what is known as Dynamic[47] Bayesian Networks (DBNs), which are a notable particular case of DGMs. DBNs have received a lot of attention for solving segmentation and classification tasks on sequences, specially for ASR. A particular case of DBNs are the so called *state observation models* which decouples the system into two parts: On the one side, the internal state of the system evolves and only depends on its previous states (and not on the current observations). On the other side, the observed variables at a given moment only depends on the current state. HMMs constitute a notable example of both DBNs and state observation models. Their graphical representation as DBNs, as observed in Figure 40, provides much insight in the underlying independence assumptions of these models: the internal state in a given slice is represented by a symbolic (or discrete) random variable whereas the observed state at a given time is represented by a fixed set of variables. Nevertheless, as better explained in Section 4.4.1 and Chapter 5, while a HMM viewed as stochastic finite state models clearly distribute the probability mass among all possible state instantiations and segment lengths, its DBN counterpart does not clearly show how the probability mass is maintained normalized since DBNs simply states that an infinite number of ground HMMs are constructed, each one for a different length.

Although HMMs constitute a particular case of DBNs, DBNs are much more general. We refer the reader to [Murphy 2002; Chapter 2] for an extensive review of how different HMM variants can be expressed as DBNs. Indeed, most works on DBNs found in the literature put the emphasis in capability of these models to represent hidden states in a distributed way, as is the case of factorial HMMs, and even to use continuous representations of the internal state as is the case of Kalman filters and, to some extent, to recurrent neural networks. These issues will be discussed into more detail in Section 4.4 when some approaches are compared with the two-stage generative process.

distributed & continuous hidden states

Time-slice templates and the unrolling process applied to BNs to obtain DBNs can be used together with other graphical model types as well. This technique is considered as *de facto* in CRFs and has been recently used with factor graphs [Mirowski and LeCun 2009] and, quite recently, even with sum product networks [Melibari *et al.* 2013].

47 The term "dynamic" is not very fortunate since it does not mean that the network changes over time, but this term has already been coined.

4.2 TWO STAGE GENERATIVE MODEL

Let us remark that the main goal of this chapter is the study of a family of doubly stochastic generative models and that the purpose of previous sections was limited to provide some background. Two stage generative models (TSGM) describe a probability distribution over pairs composed of a finite length sequence of labels or "latent units" and an observed finite sequence of labels, frames or feature vectors[48] which is *segmented* into the same number of segments as in the first sequence. This corresponds to the generative factorization described in previous section. It is usual to consider these pairs as if they were generated in two stages, hence the name we have adopted for them, as illustrated in Figure 41:

- a sequence of hidden[49] units or labels is generated;
- each unit from this hidden sequence generates (or explains, or is associated to) a segment of the observed segmented signal. The correspondence between latent variables and segments is monotone. Reordering and overlapping of segments in the observed signal are not considered.

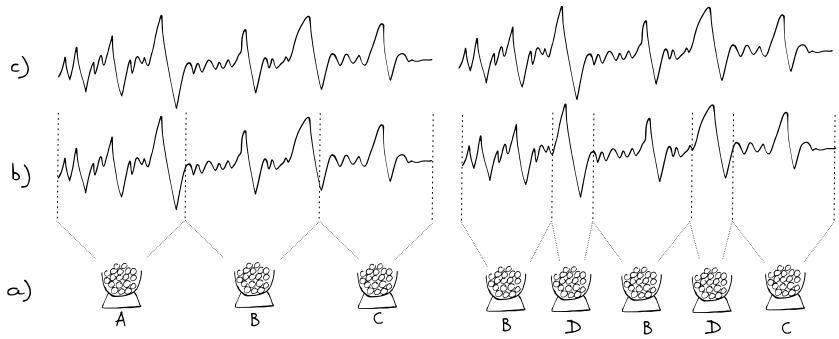


Figure 41: Two stage generative model: **a)** a sequence of hidden labels (depicted as "urn types" in the figure) is produced in a first stage, **b)** a segment is generated from each urn, which produces a segmented signal. **c)** In practice, the segmentation information is not available and the same observed signal may have different explanations which give support, in a probabilistic framework, to the same evidence (left and right examples are identical).

The second stage, which roughly correspond to the concatenation of the realizations of each segment to obtain the completed observed sequence, reminds us the Beads-On-A-String metaphor from [Ostendorf 1999] and poses continuity problems in the neighborhood of adjacent segments as well as problems of correlation between them. It is possible to consider the existence of correlations between nearby segments, as described in Section 4.2.1. Also, the distinction between

48 A continuous signal can be approximated, in some cases, by means of a sequence of feature vectors.

49 We have followed the same designations as with HMMs, where terms "hidden" and "observed" are conventional variable designations that does not necessarily mean, in this context, that observed variables are known and hidden variables are unknown in all problems types where these models are used.

extra-segmental and intra-segmental sources of variation, when generating segments, is described in Section 7.2.2. Before deepening into these questions, let us first make some observations:

language and observation models

- the first stage is a generative model of hidden units which, in some contexts, is known as language model (LM). LMs are described into more detail in Chapter 6. The second stage (also known as "observation model") can be considered a likelihood when viewed as a function of the unit type, they are the subject of Chapter 7. This decomposition is a generative factorization of a process generating labeled segmented signals;
- in a more realistic scenario, the observed sequence is not segmented (as depicted in Figure 41 **c)**, where the same observed sequence can be explained in different ways by different hidden sequences or even by different segmentations associated to the same sequence of latent variables). However, generative models taking into account this segmentation are often easier to describe and the corresponding distribution where the observed signal is not segmented can be easily obtained by means of marginalization, if desired. Moreover, some problem types try to obtain the segmentation and some decoding algorithms which can take segmentation information into account;
- some models which fit in the previous description are also able to generate a continuous[50] stream, although the scope of this work is limited to the generation of finite length sequences/signals which are generated in a i.i.d. way.

Relating the last point, the traditional description of HMMs interleaves the generation of a latent label, given the previous one,[51] with the generation of an observed frame given that label. A continuous stream of observed frames can be obtained in this way. For the previous HMM example, a sequence of types of emission probabilities (we can think of "urns") is obtained in a first stage and a sequence of observations (like balls extracted from the corresponding urns) is generated afterwards. This alternative point of view is not at all new and has been used, for instance, in [Rabiner and Huang 1993]. We have simply tried to generalize it to a wider family of models.

Let us remark that models described here are not a direct consequence of dividing the observed signal into segments. There are models which follow similar assumptions but are not probabilistic and generative. For instance, conditional or discriminative models try to estimate the probability of latent variables conditioned on the observed signal. Non-probabilistic models (e.g. based on distances) relying on the same assumptions (sequences of latent units associated to segments of an observed signal in a monotone way) can also be found in the literature (e.g. [Aibar 1997]). For the sake of completeness, Section 4.4 is devoted to briefly review and compare some alternative approaches.

50 To avoid confusion, we are meaning here a "source" of symbols, and not the mathematical concept of continuity.

51 This unobserved label is usually described as a state, so it is more common to say "given the previous state".

source-channel paradigm

The two stage generative process is also related to the source-channel (also known as noisy-channel) paradigm [Bahl *et al.* 1983]) where the first stage represents the source and the second stage corresponds to the channel. This point of view emphasizes the idea that the information, the "message", is only conveyed by the sequence of units of the first stage. Otherwise stated, the hidden sequence suffices to explain the "meaning" (whatever it is) without also requiring to consider the specific way the observed signal is produced.[52] These models can also be considered as a particular case of "doubly stochastic models".

characterized by the interface between them

We are characterizing a *family* of models in a very general way since very few assumptions relating the first and the second stages are made: we are only restricting the *interface* between them. This is a very orthogonal approach since different models can be used independently at each stage as far as the interface between them, *a finite sequence of labels*, is respected. For instance, it is perfectly possible to use context free models (described in Chapter 5) in the first stage where most systems rely on finite state automata (see also Chapter 5). It is also possible to use general generative *segment models* (SM) other than HMMs at the second stage.[53]

But adopting too general assumptions has some limitations: additional constraints are probably required on the models of each stage to be able to describe useful decoders. The best solution would be to provide, for each decoding algorithm, the minimum set of required properties together with its description in terms of these properties in an as general and agnostic way as possible and, conversely, be able to group the set of decoding algorithms compatible with certain properties following the general ideas of organizing a joint map of techniques, tasks and features, as described in Chapter 1. Unfortunately, it is not worth the advantage/effort ratio if the study is restricted to just some particular cases, as will be the case. That is why some pragmatical simplifications will be taken. Models for the first stage will be covered in detail in Chapter 6, but we can assume that it will be possible to compute the probability of generating/observing contiguous sub-strings[54] and, more commonly, the probability of certain lexical item given the past history (the prefix, although prefixes with a similar behavior to this respect are clustered in the form of a LM *history*). On the other side, Section 4.2.1 addresses some independence assumptions relating the second stage while Chapter 7 is devoted to the study of some models for the generation of segments in this stage (they also known as "observation models", "realization models", "segment(al) models" or even "acoustic models"[55]), for a particular case of the proposed hierarchy.

52 Which is a simplification in many tasks. For example, prosody may also carry semantic information in speech.

53 SMs consider that the generation of a frame sequence is a basic step instead of being limited to emit individual frames. They are briefly described in Section 4.4 and, more extensively, in Chapter Chapter 7. Since SMs does not explicitly prescribe how the frame sequence is generated, they contain HMMs as a special case.

54 Sub-string is not the same as sub-sequence. In both cases, the order is preserved but sub-string symbols are also required to be consecutive in the original sequence.

55 For the particular case of ASR, but this term is *so* coined in the literature that we cannot avoid mentioning it! For the HTR case, some authors use the term "optical models" whereas others prefer "morphological models".

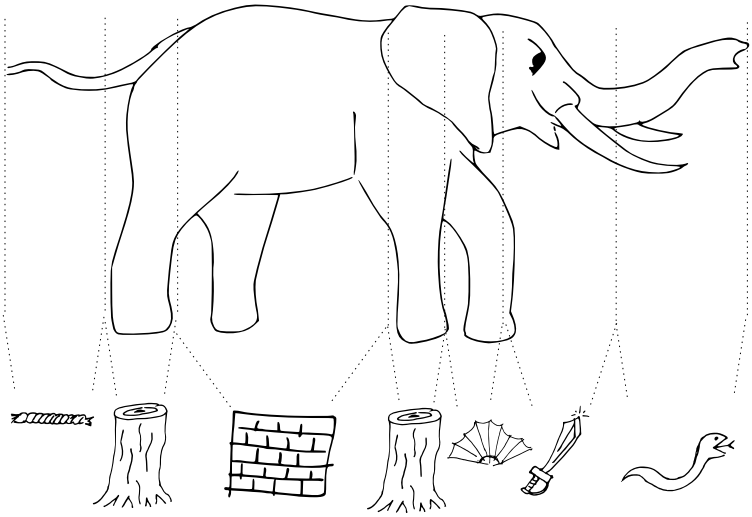


Figure 42: Metaphorical illustration of the two stage generative model inspired in poem *"The Blind Men and the Elephant"* by John Godfrey Saxe. This figure illustrates the same concepts as Figure 41.

A metaphorical example

Although we could have taken some of the examples described in Section 4.3, we have preferred to start with a metaphorical example inspired by the famous poem *"The Blind Men and the Elephant"* by John Godfrey Saxe and we have preferred to use it to depict a metaphorical example. In this story, known much before the poem, different blind men describe what they consider to be an elephant depending on the partial information they receive by the touching. In this metaphor, the entire elephant would be the observed signal, the sequence of descriptions (namely: 'rope', 'tree', 'wall', 'tree', 'fan', 'spear', and 'snake') would be the sequence of latent or hidden labels, as illustrated in Figure 42. This sequence plays the same role as the phonetic transcriptions or even the LM in ASR. The second stage of the generative model explains, in this case, how to generate the sequence of touching sensations[56] each blind men experiment for each "segment" or part of the elephant. As with the blind men from the ancient myth, the joint information obtained from different people together with the information about how these different partial pieces would fit together provides insight on the global shape.

4.2.1 Hierarchy for the second stage

It is possible to establish different hierarchies of models of the second stage depending on several criteria. In this section, two of these criteria will be considered: the allowed segments lengths and the independence between the generation of different segment (how the generation of segments is conditionally dependent or not on the value of the values of nearby labels and on the neighboring segments).

56 Sequences of "touching sensations" would correspond to extracted features.

Segment lengths

Let us see how the models of the second stage can be classified depending on the allowed set of segment lengths. The following list (from less to more general) is a crude approximation since there can be lots of possible distributions of segment lengths, note also that these distributions may depend, in the general case, on the latent labels:

- the length of a segment associated to a given type of unit is fixed, given that unit. This is the strongest assumption and, as we will see, has very important consequences relating decoding. If this assumption seems artificial at a first glance, note that it is the most common used, since HMMs are a particular case where each hidden unit emits always one frame at a time;[57]
- segment lengths can take on a finite number of values, the most common situation consisting in establishing a maximum segment length. This limit is usually an approximation for decoding performance purposes (duration pruning) which may seem reasonable because the probability mass associated to longer and longer segments must decrease towards zero;
- segments may, in principle, take an infinite number of lengths. In the most general setting, any length is allowed in principle.

explicit duration modeling

Let us remark that some segment models are described so that the segment length is *explicitly* modeled by means of a *duration model*, and a statistical model for the corresponding known duration is used afterwards (a family of statistical models parametrized by duration is usually provided, see Section 7.3). Other segment models are not described in this way and, in this case, the length is just a consequence or a "byproduct" of the overall segment generation process. For instance, the length in HMMs depends on the topology.

The previous classification is not inherently related to a given task or even to a given type of models, since it can depend on the point of view and on what we consider to be a segment: HMMs, described below, can be considered as "miniature two stage generative processes" where hidden states of HMMs are responsible of a frame which can be trivially considered "frame sequences of fixed length one". But, when we consider the same model at the word level, each segment is a frame sequence associated to a word, whose length may vary. Thus, the same model may correspond to two different cases in the previous hierarchy. This distinction is not only a question of aesthetics or a matter of taste or of philosophy, it has practical consequences as well: Due to the associativity of the composition of models, the same problem can be solved by means of two stage decoder or by means of one step decoder (to be described later), being the last one more efficient for long sequences because of the lower ambiguity during the "matching" process, which avoid the quadratic[58] cost associated to it.

57 Most successful approaches to ASR and HTR are based on HMMs. One-pass decoders are based on this assumption. HMM states not emitting frames can be considered an artifact equivalent to null transitions in finite state models.

58 It is quadratic with the input length when the realization of different segments does not depend on the realization of others (only on underlying labels) and when they can be incrementally computed.

Context dependency

Another criterion to classify the models of the second stage is the correlation between different realizations of nearby units of the hidden sequence. Before considering how this dependency can be tackled by models and decoders, we should first try to characterized and to organize some types of local dependencies not only with nearby segments but also with nearby hidden labels from the first stage. Indeed, the limitation of dependencies to a local neighborhood seems a reasonable assumption for many tasks, since the second stage is usually associated to a physical phenomena. In our opinion, the following parameters[59] seems a reasonable way to characterize the context dependencies in the two-stage generative process:

reasonable assumption

- '**l**' measures a number of hidden labels. It is the distance to neighboring labels such that the realization of a segment associated to the current hidden label, conditioned on neighboring labels at this distance is the same as conditioned on a larger context;
- '**s**' measures a number of segments in the observed signal. The realization of a segment depends on the duration of nearby segments but not necessarily on their particular realization;
- '**r**' the realization of a segment associated to a given label depends on the '**r**'ealization (or particular shape) of the segments associated to neighboring labels up to a distance of length r, which is a much stricter constraint than previous parameter;
- '**f**' is related to the frame emission regardless of the use of segment models. This parameter is used to indicate that the emitted frame depends on a maximum (left, right or both) context or f neighboring frames.

These parameters can take different values in the left and in right contexts. For instance, $\mathbf{s}$ can be distinguished into $\mathbf{s}_{\mathtt{left}}$ and $\mathbf{s}_{\mathtt{right}}$ when they are different. When not distinguished, we assume the maximum of both. There may exists more fine grained parameters such as a dependence on durations or realizations limited to some particular combinations of hidden labels.

Let us make some observations on these parameters. The dependence of '**l**' parameter can be removed by replacing the hidden labels by context dependent units (e.g. triphones in Speech). This relabeling process can be considered as an intermediate stage so that the first stage (sequence generation) is not affected (see Sections 6.9.1 and 7.6). The fact that some models of a hierarchy can be replaced or at least approximated by others from a simpler type (at the expense of more parameters) does not seem unusual or strange.

context dependent units

Segmentation constraints and the corresponding parameter '**s**' are interesting when using segment models with explicit duration modeling, although all explicit duration models found in the literature are assumed to be independent of durations of neighboring segments. This parameter seems more tractable than '**r**' because the search space remains finite and can be constructed a priori, for a given observed length, as an extension of the "two stage" algorithm. However, the

59 Their names are not standard because we have not found them in the literature.

search space explodes (see Section 4.3). There is a clear relationship between '**s**' and '**r**' since we can ignore the first one when the second one is greater or equal (dominates) the second.

overlapped sliding window

It is not strange to observe a correlation between neighboring frames in tasks where the parametrization stage is based on sliding windows. In somr ASR parametrizers the windows may be partially overlapped up to 50% with previous frame. Another contribution to frame correlation is the use of augmented features in HMMs, which has been common in ASR for decades.[60] Although some models try to take the restrictions of augmented features explicitly into account [Zen *et al.* 2007], most other approaches simply ignore the mathematical correctness and use them without other justification than pragmatical considerations: their use consistently increases the recognition performance.[61] This means that even if '**f**' parameter might make some sense, it is not necessarily taken into account *as is* by most decoders.[62]

If it is easy to justify the practical interest of '**l**' parameter (by relating it to context dependent units which are widely applied in many tasks where they have proved useful) and the relevance of parameter '**f**' (despite the lack of a proper mathematical treatment in most works). It seems more difficult to justify the practical relevance of '**s**' and '**r**' parameters. Indeed, we have not found any work in the literature describing (not to mention proposing) their use. We can guess that this is because these relationships does not appear in many practical tasks and, where they appear, it would be not worth modeling them the cost of decoding becoming prohibitive[63] since they produce a much larger increase of the search space . Nevertheless, let us point out some examples in order to succinctly illustrate some hypothetical examples of their use: on the one side, [Shi *et al.* 2007] proposes a semi-Markov structure for automatic paragraph segmentation where each boundary depends on the position of two of its neighbors (see Figure 43) which seems related to '**s**' parameter. On the other side, Figure 44 illustrates the interest in modeling relationships between nearby segments by means of concatenation costs, which seems related to '**r**' parameter even if it does not correspond to a probabilistic approach.

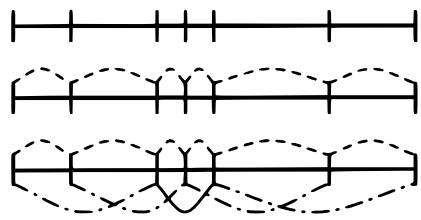


Figure 43: Figure taken from [Shi *et al.* 2007] to illustrate the interest in '**s**' parameter: a semi-Markov structure is proposed for automatic paragraph segmentation where each boundary depends on the position of two of its neighbors.

60 Cepstral coefficients together with their first and second derivatives.

61 But see [Bridle 2004] for an interesting point of view on how the use of HMM with dynamic features defines an unnormalized generative model.

62 Hybrid HMM/ANNs (Section 7.5.3) have a parameter which slightly resembles '**f**': the number of neighboring frames taken into account at the input of the ANN when classifying the posterior probability of each HMM emission type given the frame.

63 It could be used for reraking in a multi-pass architecture, instead.

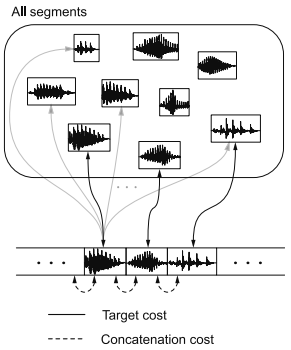


Figure 44: Scheme of a unit-selection process for speech synthesis (taken from [Zen *et al.* 2009]). The relationship between neighboring segments is represented, in this case, by means of concatenation costs.

4.2.2 Some limitations, extensions or possible generalizations

Two-stage generative models have several limitations, let us see some of them. We believe that they are better understood in the context of a reasonable trade-off between usefulness and generality:

MONOTONY refers to the fact that observations obtained from the sequence of hidden labels are generated in a label-by-label basis without the possibility of reordering. This clearly limits the applicability of these models to some tasks. For instance, we saw in Chapter 2 that SMT requires reordering in the general case. The proposed model can be applicable, at least, to monotone machine translation. Nevertheless, some decoding techniques described in this work are useful for some monotone SMT systems and can even be reasonably adapted to take reordering into account.

SEQUENCES OF DISCRETE LABELS meaning that the sequence of labels or hidden units is composed of symbols from a finite alphabet. There are other models based on continuous hidden state variables such as Kalman filters or recurrent neural networks, to mention two examples. This feature does not limit the possibility of applying these models to continuous signals generated in the second stage provided that they correspond to a finite repertoire of hidden symbols (phonemes, graphemes, etc.). Let us remark that this work is limited to the study of discrete time systems. Indeed, continuous sequences have to be discretized unless we deal with analytic representations of continuous functions instead of data being captured by sensors. An example is ASR where, as observed in Section 2.8, the use of HMMs is justified by the assumption that the rate at which frames are sampled is high enough to preserve the properties of speech that convey the message. Relating the continuity of the signal at the segment frontiers, there is a clear modeling limitation when assuming that neighboring segments are generated independently.

Let us finally remark that the use of a finite repertoire in the hidden labels (the labels of the interface between stages) seems not impossible to overcome: some generative models suitable for the first stage may use a continuous representation of labels, as is the case of neural network language models (NN LMs). Continuous

and distributed representations may improve the estimation capabilities of these models by making "similar" labels to have a closer representation. This issue will be discussed into more detail in Section 6.3.4. Although it is possible, in principle, to use this type of continuous representation as an interface to drive the second stage, we will assume in this work a finite repertoire of hidden labels, mainly when dealing with decoding.

A SEGMENTATION (NON OVERLAPPED SEGMENTS) meaning that segments obtained in the second stage are concatenated to produce the (segmented) observed signal. Segments from a same outcome constitute a segmentation, they are not overlapped. If we are not interested in the segmentation it is possible to marginalize the segmented signals over all possible segmentations. Remark that, even for a given sequence of hidden labels, different segmentations may contribute to a given hypothesis. Conversely, the most probable segmentation may take into account different sequences of hidden labels. However, the most common approximation used in decoding (the Viterbi approximation) computes the best joint segmentation and classification and neither the sequence of hidden labels nor its associated segmentation are necessarily the most probable ones.

MULTIPLE FEATURES to model the observed signal is discussed in many parts of this work. Basically, two different approaches can be distinguished:

- a sole sequence of frames is obtained where each frame is obtained by concatenating different parameters;[64]
- each parametrization produces a different sequence of frames which does not necessarily have to be synchronized among them.[65] This approach seems more flexible since each feature might have a different frame rate.

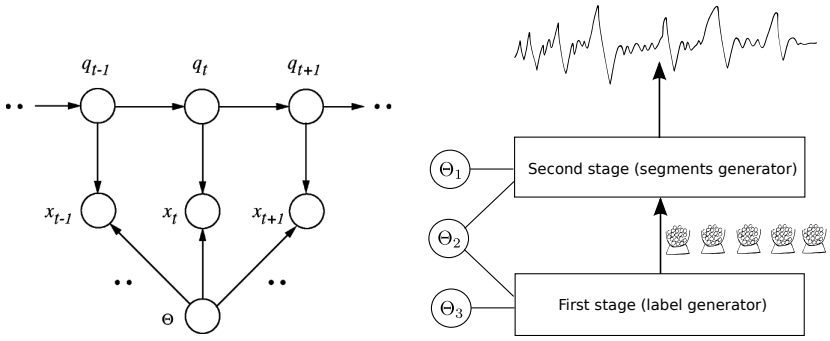
The first possibility does not require a special treatment and usually affects the emission of frames in the second stage.[66] The second one is more related to *multi-tape* models which will be discussed in other sections of this work and, in particular, in Chapter 5 where parsing is described using semirings, which are a suitable way to describe this issue.

CORRELATION BETWEEN BOTH STAGES the idea of using a sequence of hidden labels as the sole interface between the first and the second stage of the generative model seems attractive due to its simplicity, but it has some limitations as illustrated by the following ASR example: certain features such as the rate of speech or the level of spontaneity may affect *both* the use of certain words and the acoustic models. Let us notice that the rate of speech is

64 This approach is called "multi-source" in some works.

65 It is possible, in some cases, to add some synchronization constraints as described in [Bourlard and Dupont 1997].

66 In particular, two strategies can be mentioned, namely: either to train an emission model for each parameter in order to combine them afterwards, or to train a unique estimation model (which may involve applying first some de-correlation techniques to the concatenated frames).



(a) Bayes network representation of a parametric HMM.

(b) Parametric 2-stage generative model.

Figure 45: Parametric models. In **(a)** (taken from [Wilson and Bobick 1999; Fig. 2]) random variable Θ influences the emission of observed variables x_i. In **(b)** three random variables can be observed: Θ_1 plays the same role of Θ from left figure, Θ_3 would play a similar role in the first stage whereas Θ_2 is a possible way to express additional correlation between both stages.

not associated to any particular or individual hidden label, frame or segment but is rather shared by all of them across the entire utterance,[67] so it is possible to consider it as a unique or global random variable in the model, as is described in [Wilson and Bobick 1999] (in this case, influencing the observed frames) and depicted in Figure 45 **(a)**.

These type of parameters can be included either in the first, in the second or in both stages, as shown in Figure 45 **(b)**. An example for the first stage (random variable Θ_3, in the figure) is topic adaptation which can be implemented by weighting a set of LMs (as discussed in Section 6.5). An example for the second stage (random variable Θ_1, in the figure) could be vocal tract normalization. The particularity of the rate of speech is that it can influence both stages. This additional source of correlation can be included as an additional value in the interface between both stages, as illustrated in Figure 45 with random variable Θ_2. A possible way to deal with this feature is to estimate the value beforehand by means of some technique "external" to the decoding process. Since this stacked approach makes it difficult to recover mistakes, different alternatives could be taken into account in order to deal with ambiguities. The presence of alternatives is translated into several decoding processes, but this is still suboptimal since a finite and limited set of parameters have to be

67 We are considering here the simpler case where the rate of speech is variable among different utterances but almost constant for a given one. The same consideration is usually assumed for the vocal tract length and even for the slant in handwriting, but note that there are other possibilities such as non-uniform slant correction, as described in Section 14.4.1. Also, some works such as [Miguel *et al.* 2008] describe decoding techniques for vocal tract normalization with smooth variations of this parameter in the same utterance. [Stephenson *et al.* 2004] also considers these "auxiliary features" to be slowly changing. Since these features are not dependent on the state sequence and seems to be estimated by means of external techniques, they are used as additional input or "observed" variables.

chosen. Nevertheless, since only the best result is required, it is worth trying to avoid some computation by combining intermediate results from these decoding processes. This constitutes an interesting line of research for extending decoding algorithms. Iterative techniques can also be envisaged.

Note that external variables do not suffice to deal with variable-parameter extensions when these values may change [Yu *et al.* 2009].

TWO STAGES Various phenomena occur at different scales in speech and, consequently, the decomposition into two stages can be placed at different levels: A speech utterance can be segmented into words, a word can be segmented into sub-word units (such as triphonemes,...), and sub-word segments may be decomposed into frames. So, why do we consider only two stages? We believe that this is the simpler model suitable to describe the decoding algorithms studied in this work.

Obviously, models can be stacked so that the model associated to a segment from one model is another two-stage model: A phonetic unit based on HMM can be considered as miniature two-stage generative model. This is not different from the transducer composition technique discussed into more detail in several parts of this work (e.g. Chapter 5). However, as expected, reality is more complex and this stacked approach exhibits some limitations since it is difficult to deal with properties *halfway between different levels*. For instance, certain phonetic properties depend on the relative position of the phoneme inside the syllable.[68] A similar phenomenon can be observed, at the word level, with word boundary effects (such as french "Liaison" and other types of external sandhi described in Section 2.8).

Decoding algorithms which are able to deal with these phenomena should hide these complexities as much as possible. It would be desirable, for instance, to generate a DAG of phonemes labeled just with context independent phonetic labels even if context dependent units are used. In this way, the following stages that may receive these DAGs as input does not need to be aware of the additional complexity.[69]

Besides the use of stacked models with a predetermined number of levels, where each level has some specific properties, we can also imagine multi-stage hierarchical systems described in a recursive way which are able to deal with an unbounded number of levels. This is related with the use of context free models addressed in Chapter 5. To this regard, we believe that, for most applications, this unbounded hierarchy can be addressed in the first stage so that the use of the two stage generative model remains valid for them.

68 This, as well as the use of some type of context dependent units based on the second half of a unit tied with the next half of the next one, discussed in Section 7.6, illustrate some (apparent?) limitations of the '1' parameter proposed before.

69 Nevertheless, this feature can also be modeled by means of context-dependency finite state transducers.

4.3 CLASSICAL PROBLEMS OF TWO STAGE GENERATIVE MODELS

coined expression

The "three classical problems" as described, for instance, in [Rabiner and Juang 1986; Secion C] is a coined expression concerning HMMs. Nevertheless, they can also be extended to other model types such as GMs, described before, as well as to TSGMs. These problems can be stated as follows:

- **probability of observation:** given a model and an observation, compute the probability of generating this observation;
- **decoding:** given an observation, which are the hidden labels which best explain it? and
- **(supervised) estimation** from a set of labeled observations.

variants

It is also clear that these problem types admit variants. For instance, the estimation problem may take into account more or less information leading to unsupervised, semi-supervised and even active learning approaches. It is possible to just adapt an already trained model (model adaptation) and so on. Decoding may generate several explanations and the labels may be aligned or not with the observed data, etc. Let us also remark that there are other problems related to them (some of them were described in Chapter 3). An example of a related problem is:

related problems

- **synthesis:** given a model and a label sequence, obtain an admissible observation associated to this sequence (e.g. speech synthesis).

Before entering these classical problems, we would like to make some observations. Firstly, note that these problem types can be a piece of another (possibly different) problem:

- model estimation is usually performed by means of ascent gradient techniques or growth transformations which may require alignments that are closely related to decoding. Some discriminative techniques may require the generation of the best hypotheses;
- some recognition systems rely on multi-pass approaches. For these reasons, we prefer to describe the basic pieces, principles and general ideas. Arbitrary combinations or compositions of these systems could be conceived withouth much effort afterwards.

parametrized by additional restrictions

Secondly, note that, when studying TSGMs, we are dealing with a family of models mostly *specified by a restriction*. This seems a too abstract level of detail that we risk to be able to say nothing at all. Indeed, most (even general) algorithms require to consider some additional restrictions. A complete approach would imply to parameterize the proposed algorithms depending on these additional restrictions but, for the sake of brevity, only the more general assumptions will be considered. This concision is justified because entire chapters will be devoted to this matter through the rest of this work. Besides, since this work is mainly limited to decoding, we will not deep into model estimation.

4.3.1 Probability of observation

The most common inference problem is the calculation of the likelihood of the observed evidence given the model. The brute force approach would require to take into account all possible underlying sequences up to a fixed length[70] and, for each one, to sum up all the ways of segmenting the signal into the proper number of segments.

running the generator in reversed way

This naive approach is based on running the generative model *in a reversed way* by using non determinism when some information is lost during the generation (in particular, when segment boundaries are removed). It is difficult to describe more efficient ways to tackle this problem at this broad or coarse level of detail. Of course, it is not practical since both the number of label sequences and the number of segmentations grow exponentially with the sequence length. It is therefore required to consider some additional assumptions at each stage of the TSGM. Since we are running in a reversed way, let us first address the second stage.

Relating the second stage

Our point of departure is a frame sequence with n frames, which contains $n+1$ frame boundaries. It is required to take into account the independence assumptions between the realizations of nearby segments described in Section 4.2.1. The most permissive hypotheses would assume that the probability of observing a segment, given a hidden label, does not depend on the particular realizations of nearby segments and would not depend on the values of nearby labels either. The different number of required segment estimations, under this assumption, corresponds to the binomial coefficient indexed by $n+1$ and 2:

$$\binom{n+1}{2} = \frac{(n+1)n}{2} \in O(n^2)$$

This assumption makes the problem more tractable since it implies that the number of segment estimations only grows quadratically with the length of the observed signal.

The set of probabilities of observing each of these segments can be efficiently described in the form of a DAG. This representation is at the core of the two-pass decoding algorithms and has the particularity that edges are labeled with likelihoods and vertices are associated to positions at the observed signal. It can also be seen as a compact representation of a finite weighted language.[71] These formal languages are described in Chapter 5.

The use of a DAG for compactly describing how a sequence can be decomposed into a set of segments is not at all new and, indeed, predates the era of probabilistic models when distance based methods were the common practice [Sakoe 1979; Hiroaki 1982]. The most common models used to estimate the likelihood of observing a segment

70 Assuming that no hidden sequence might generate a zero length segment.

71 Not to confuse with an stochastic model because, here, there is no contraint relating the sum of probabilities.

are reviewed in Chapter 7. Some algorithms for computing these likelihoods are described in Chapter 10 and the generation of the DAG in an online way suitable for using it while it is being generated is the focus of Chapter 11.

It will be clear through all these chapters that the DAG structure is related with the transitive closure of a former DAG constituted by the linear concatenation of the frame sequence where vertices are also frame boundaries and where DAG edges are labeled by frames. Let us now study the influence of parameters described in Section 4.2.1:

transitive closure

- '**l**' means that the realization of the segment also depends on a given neighborhood of hidden labels. DAGs generated when using those context-dependent units require to associate several vertices or states to the same segment boundary in order to prevent the concatenation of mismatched segments (because the context of adjacent segments are related by their contexts). In practice, contexts dependent units are tied and their number is reduced w.r.t. the set of possible combinations. The size of the DAG may grow substantially, but usually remains tractable;

- '**s**' means that a segment is also characterized the duration of nearby segments up to a given left and right distance. The number of these edges is related[72] to the binomial coefficient indexed by $n+1$ and by $s_{\mathtt{left}}+2+s_{\mathtt{right}}$ (where context length s can be distinguished into $s_{\mathtt{right}}$ and $s_{\mathtt{right}}$), which is usually intractable even for low values of '**s**';

- '**r**' is similar to '**s**' but the particular shape of neighboring segments is also taken into account. This difference has no effect, in decoding, on the number of segments unless we are already using a DAG of frames as a point of departure (e.g. for ambiguous front-ends, see Section 2.9.3). In any case, this extension severely affects the underlying models used to estimate how likely is observing the segment;

- '**f**' is related to the neighborhood measured in a number of frames regardless of segment boundaries. Again, this value has no effect on the DAG size unless departing from a DAG of frames.

Only the extension for context-dependent units has a prospect of success. The others will not be further investigated in this work because we do not know problems for which the possible benefit of these extensions justifies, in practice, the associated increase of the computational cost.[73]

It is also possible to use task-dependent information to impose segmentation contraints on the generation of these DAGs in the form of maximal segment length, oversegmentation points and so on.

72 Indeed, this value is approximate since it does not take into account the problems at the boundaries of the signal.

73 The insightful reader may wonder why this hierarchy has been proposed in the first place at all. A possible answer is that we are not limited to the most practical realm of engineering but also to mathematics.

Relating the first stage

The first stage is related to models describing the a priori probabilities of sequences of hidden labels. This is known as language modeling (LM) and is the subject of Chapter 6. For the moment, it suffices to observe that, when the LM is represented by a finite state model (described in Chapter 5), there is an efficient and well known algorithm, known as *forward algorithm*, to compute the desired summation by means of dynamic programming. The extension of this algorithm for context free languages is known as *inside algorithm*, the relationship among both algorithms is addressed, again, in Chapter 5.

In the particular case of using HMMs and LMs based on finite state transducers, this algorithm can proceed in a frame-by-frame basis. Some works have proposed the extension of the forward algorithm to the case of segment models although, as will be sharply clear in this work, these extensions are just the application of the same simple algorithm which is a particular case of transducer composition.

illusory distinction

This illusory distinction has its counterpart in decoding where two-stage and one-pass algorithms describe the corresponding approaches based either on processing a DAG or on advancing frame-by-frame, respectively.

Indeed, in order to easily show that one-pass is a particular case of two-stage, it suffices to observe two things: firstly, the fact that a DAG is constructed to be combined with the LM does not mean that it has to be completely constructed in advance. DAG construction and LM combination can be performed at the same time. Secondly, one-pass arises form the fact that it admits the following interpretation: only length one segments are used, so that the DAG no longer has the size of the transitive closure. Indeed, it remains the linear sequence of frames.

4.3.2 Decoding

Decoding is usually associated to the problem of identifying the most likely sequence of latent labels given the observed evidence. This is a particular case of the Bayes optimal decision for a particular loss function which measures the sentence error rate (SER). Nevertheless, there is a more general setting where the sequence of latent variables minimizing the expected loss, for a given loss function, is computed. Since, at least in ASR, the quality of the recognition is usually more related to the number of words the user has to manually correct (which is measured by the word error rate (WER)) than to the number of perfectly recognized sentences (SER), there is an interest in Minimum Bayes risk decoders. Nevertheless, let us focus for the moment on computing the sequence maximizing the probability of observation.

As in the previous case of computing the probability of observation, we can also use the metaphor of running the generative model in a reversed way. It is also required to assume some independence assumptions between the realization of nearby segments in order to obtain the same DAG representing the likelihoods of observing each hidden label at each segment. The same remarks about the effect of the independence assumptions on DAG size apply here. Likewise, the

LMs for the first stage are similar as in the previous problem type. What differs is *how* these information sources are combined.

For the case of LMs represented with finite state models, it is possible to compute, by means of dynamic programming, the path of highest probability among the paths of hidden states aligned with a particular segmentation of the observed signal. The sequence of hidden labels associated to this solution is usually known as Viterbi approximation because it does not necessarily correspond to the sequence which best explain the observed signal. Similarly, the obtained segmentation does not correspond with the best segmentation either.

In order to obtain an "optimal decoding" it would be required to take into account the fact that several paths in a finite state model can be associated to the same hidden label sequence and, likewise, that this sequence can be associated to several segmentations of the observed signal. Unfortunately, it has been proved that optimal decoding is an NP-complete problem [Casacuberta and de la Higuera 1999]. This is the reason why approximations, such as the former Viterbi approximation, are used instead.[74]

optimal decoding is intractable

The usual algorithms to tackle the first two "classical problems" are very related. In both cases we have the same elements and they only differ, apparently, in how they are combined. But there is also a perfect correspondence between forward and Viterbi algorithms. As will be explained in Chapter 5, they are just two instances of the same algorithm described in terms of semirings. Only the used semiring type is what differs.[75] This is not at all surprising since this is just what happens with GMs where the same problems are also particular instances of a summary-product algorithm. Similarly, the models where forward and Viterbi apply are particular instances of DGMs.

max vs sum

4.3.3 Model estimation

The classical estimation problem from [Rabiner and Juang 1986] is usually focused on learning the parameters of a HMM from a set of observations. The HMM topology is fixed and HMM emissions are assumed to be from a given parametric family.

The estimation of models is a little beyond the scope of this work. since we have been more concerned with decoding. We basically refer the reader to the general estimation methods described in Section 4.1 and to some papers and surveys such as [Deng and Li 2013; and references therein].

We plan to address model estimation as a future work and our main aim will be to factorize the training algorithm so that each part is as much orthogonal as possible and to provide a common interface for the large plethora of segment models described in Chapter 7.

74 Nevertheless, relating the optimal decoding and the use of other loss functions, the following particularity observed in practice constitute a great help: the set of best hypotheses, which can be obtained in a quite efficient way, accumulate a great part of the probability mass. It is therefore possible to use multipass techniques to obtain accurate solutions to these difficult problems.

75 However, this difference has a lot of consequences regarding optimization. Particularly, the Viterbi approximation leads to shortest path problem in the more classical sense where the result comes from a sole path, not in the more general definition of shortest paths used in Chapter 5.

4.4 SOME ALTERNATIVE MODELS

It is convenient to take a brief look at some alternative models proposed in the literature for the sake of completeness and also to put into context the family of models we are describing. The following list is not exhaustive nor mutually exclusive as illustrated by the following examples: TSGMs are highly overlapped with DGMs, some discriminative models which can also be considered particular cases of DGMs and most fixed dimensional segment models are discriminative, to mention a few.

procedural descriptions are left aside

This section has left aside procedural descriptions or "recognition recipes" and the focus has been on probabilistic (both generative and discriminative) models. Nevertheless, we have devoted a little section to describe a particular instance of an energy-based system (graph transformer networks) and other to distance based systems.

4.4.1 Relationship with Dynamic Graphical Models

questions on unrolling

Although DGMs described in Section 4.1.1 may cover quite different types of models, we will limit our comparison to 2-time slice DBNs where a template is unrolled in a linear way as times as required to fit the length of an observed sequence. In this way, each unrolled DBN may model paths of a fixed length in a (not necessarily finite) state model: the values of the variables in each time slice represent the corresponding state in this position, where it is clear that a possibly structured and factorized representation of the state is possible by using several random variables. Finally, transition probabilities are modeled as dependencies between the value of these random variables and the corresponding variables in the next slice.

It is clear from Section 4.1.1 that GMs can be used to model both generative and discriminative models for a predetermined set of random variables. In the case of DBNs, we can say that they represent a family of these ground GMs (the unrolled ones), one for each possible length. Unfortunately, this does not constitute a "generative model on sequences" in the same way as, for instance, a stochastic finite state model (SFSMs, see Chapter 5). The difference is clear when we consider that in a SFSM there is a probability mass that is distributed among all possible sequences, no matter their length. However, in the case of DBNs, each length is modeled by a different ground GM and there is not equivalent of "transition probability" to specify that a given length is assigned less probability mass. In this way, it could be possible to assign the unit mass for each possible sequence length and, hence, to represent a non proper distribution. This issue also appears in Chapter 6 when discussing the generation of sequence of labels where, in order to ensure a proper distribution, a distinguished final label "end of sentence" (<EOS>) is considered at the end of the sequence to avoid state sequences finishing in a non-final state.[76]

76 Special symbols or labels at the beginning and at the end are also known as "context cues". Without taking <EOS> into account, the sequences of a given length may be assigned the unit mass. The same idea of using a special symbol to denote the end of sequence is also proposed to the case of DBNs in [Zweig 1998; Section 6.1.2] where a random variable in the template is used to indicate the end of the sentence.

This does not mean at all that DBNs cannot be used as generative models: it is possible to emulate the transition probabilities of SFSMs and it is possible to include the constraint that state sequences must finish in a final state. Unfortunately, since this mechanism is not coupled to the unrolling process, a "trick" is required: although sequences of a fixed length which does not finish in a final state can be generated, they are assigned a zero probability.

Another question related to the unrolling process in 2-time slice DBNs is the fact that a unique chunk template is used to be repeated for each observed frame.[77] This seems reasonable when describing a simple system evolving in time as is the case of linear HMM topologies (see Section 7.3.1) but, when modeling some tasks as TSGMs, it would be desirable to change from one type of segment to another one (e.g. to change from one phonetic unit to another one in ASR tasks). This issue, known as *submodel composition*, basically consists in describing a model where some parts are also models so that the overall description is more efficient than *naively expanding every occurrence of nested models*. This efficiency is also one of the aims of TSGMs. Submodel composition is not only related to representation but also to decoding, as is well expressed in Section 6.1.2 from [Zweig 1998], titled *"why model composition with DBNs is difficult"*, which addresses the question of how to concatenate sub-models when the segmentation is not available:

invariant topology

submodel composition

> *Therefore, all reasonable partitionings of an observation sequence between the models must be considered, and possibly weighted according to some distribution. Since the number of partitionings grows exponentially with the number of sub-models, this is a demanding task that must be solved in an efficient way.*

The simple method proposed in [Zweig 1998] to describe different types of segments with a topology that has to be maintained invariant during different time instances consists in using an explicit "submodel index" random variable to specify which submodel is used at each point of the sequence. Let us observe that this value has to be repeated for each chunk associated to the frame of the same segment and has to be change in accordance with the lexicon and the language model knowledge sources afterwards. This forces all types of segment models to be represented in the same way and with a frame-centric point of view. This approach is widely used for different purposes to the extent that the standard way for representing recognition systems using DBNs (e.g. see [Bilmes and Bartels 2005]) includes several *layers*[78] of random variables to represent the word history, the position in a tree lexicon, the current phone and acoustic features... hence emulating the *internal state* of a standard ASR system. Most of these values only change under certain circumstances (e.g. word history only changes after a word transition).

frame-centric point of view

Relating the efficiency of model composition, some GM extensions are often required: For instance, the GMTK toolkit include the *switch-*

77 The existence of optional prologue/preamble/initial and epilogue/final templates does not change the situation.

78 Each variable in the chunk template is depicted in a different vertical position whereas templates are unrolled in horizontal, so that the term "layer" is, at the same time, visual and intuitive.

ing parents functionality [Bilmes and Zweig 2002; Section 3.4] which allows to select some of the parents of a given variable (conditional parents) conditioned on the current values of other parents (switching parents). This mechanism seems quite similar to the concept of *gates* [Minka and Winn 2008], described in Section 4.1.1, used to specify the context specific independence. This type of constructs may be critical for the efficiency in many cases as exemplified by the fact that the direct application of standard GMs for context free parsing (see Chapter 5) is exponential[79] but can be performed in cubic time using case-factor diagrams[80] [McAllester *et al.* 2004]. Current approaches to model context free languages with GMs essentially try to mimic parse charts from classical parsing algorithms and, hence, are not directly suitable for 2-time slicing DBNs.

Other extensions of DBNs suitable for the joint segmentation and classification on sequences include the possibility of inter-slice edges in backward direction or templates spanning multiple frames [Bilmes and Bartels 2005]. In a recent review of the most sensible techniques for the efficient use of DBNs in ASR and similar tasks, [Bilmes 2010] proposes the use of inference procedures based on the template specifications instead of applying them on the unrolled DBNs. These procedures leads to junction trees which can also be described by connecting slices. This allows the resulting variable elimination algorithm to use an elimination ordering compatible with the slice ordering.[81] It is possible, in this way, to obtain quite efficient decoders incorporating techniques commonly found in standard HMM-based decoders (e.g. pruning techniques, see Section 10.6.1) but not described in previous DBNs solutions.[82]

Let us try to clarify the relationship between TSGMs and DBNs. Not surprisingly, TSGMs are vastly less general than DBNs in many ways and, in particular, they are not suitable for representing systems evolving in time with inputs at their disposal (e.g. IOHMMs), conditional models (e.g. MEMMs) or systems where the next hidden label may also depend on past observations. On the other side, the fact that we are doing very few assumptions on the way the first and the second stages of TSGMs behave (basically, they have to be compatible with a quite general "interface" between them) leads to more generality in other aspects (as illustrated in Figure 46).

few assumptions

As we will see in Chapter 6, there exist broader classes of LMs, such as context free languages, whose incorporation into GMs seems incompatible with 2-time slicing techniques.[83]

79 According to [McAllester *et al.* 2004] but also according to [Sato 2007] when quoting [Pynadath and Wellman 1998].

80 We have not reviewed this GM type in this work. Gates seem to subsume both switching parents and case factor diagrams [Minka and Winn 2008].

81 Left-to-right or right-to-left does not matter, the important issue is that variables from one slice are chosen to be eliminated before any variable of the next one.

82 It is not unusual to observe that techniques already used for HMM decoding (pruning, lexicon tree, token passing, etc.) have been adapted to other approaches such as recurrent neural networks (see [Fischer 2012; Section 6.3] and Section 4.4.3), graph transformer networks (see Section 4.4.4) or, in this case, generalized to DBNs.

83 As described before, most techniques for dealing with context free grammars with graphical models are based on creating a model of the parse tree. Probably other type of DGM or extension could be devised to deal with context free models.

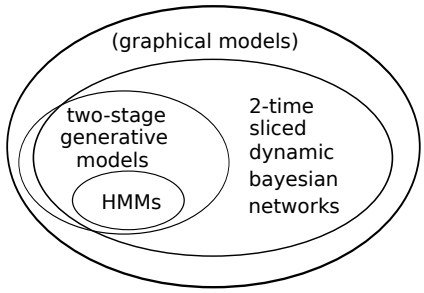


Figure 46: 2-time slice DBNs are more general than two-stage generative models in many aspects. Two stage generative models may have a portion outside 2-time slice DBNs due to the general assumptions about the distribution of sequences of hidden/latent labels. HMMs lie in the intersection.

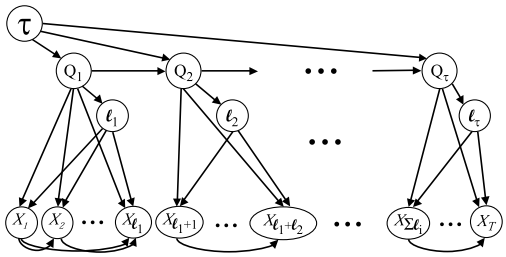


Figure 47: Figure from [Bilmes 2004; Fig 23] illustrating how a segment model can be viewed as a GM.

Relating the hierarchy of Section 4.2.1, it is very cumbersome[84] to represent, by means of DBNs, the dependence between the generation of a segment and the way nearby segments have been generated, since chunks are repeated frame-wise and variables between non contiguous slices are not connected directly.

The proposed solutions for these cases consists in replicating the previous and next labels into each slice by including more layers and by using inter-slice edges in backward direction, which is a non-standard extension described before.

Another controversial issue is the use of segment models. Although the representation of segment models by means of GMs has been proposed in the literature (e.g. [Murphy 2002; 2.3.14]), these approaches, depicted in Figure 47 cannot be used as is with DBNs but have to be emulated using layers of random variables as with other features described before. More interestingly, some particular types of segment models, such as the combination of discriminative and generative models (to be described in Section 7.7), require to take all the frames from the same segment into account in a way that seems not compatible with the representation in DBNs distributed across frames.

We believe that, although DBNs offer several advantages (many of them shared with other GM types), it seems wise to consider alternative approaches and paradigms as well. For instance, transducers (see Chapter 5) seem more suitable for dealing with model composition than the use of layers in DBNs. In a similar way, TSGMs may offer some insights and advantages that make them worth studying.

84 DBNs may indeed offer several advantages such as the use of quite general inference algorithms implemented in toolkits which allows researchers to easily experiment new ideas. However, just a claim such as "it is possible to represent such feature" by itself does not distinguish a practical thing from a Turing tarp-it `http://en.wikipedia.org/wiki/Turing_tarpit`.

4.4.2 Fixed dimension feature segments

This section addresses the segment based approaches based on fixed dimension feature segments as described in Section 2.9.1. Careful attention must be paid to avoid nomenclature confusions since the term "segment based" is highly overloaded. For some authors such as [Ostendorf *et al.* 1996] this term is associated to models capable of generating several frames at the same time from a given hidden state (as oppossed to HMMs where only one observed frame is obtained). For others, such as [Glass 2003], only the segment models discussed here in this section *deserve the right* to be called segment models, since other segment models, which ultimately produce frame sequences, are not truly segment- but frame-based. An alternative nomenclature propose the terms "frame based observation" vs "graph based observations", although we prefer to replace the last expression by "segment-based observations". Note that graph based observation systems may take a frame sequence as a starting point and what makes them "graph based" is the production of a fixed size feature vector for each segment no matter its length. Unfortunately, both types of segment model are lumped together in many works. Indeed, segment models described in [Ostendorf *et al.* 1996] are, in certain aspects, probably closer to HMMs than to "graph observation" segment models.

no matter its length

An important motivation behind *both* types of segment models (as well as the motivation of many other proposals) is to overcome the "frame independence assumption" of standard HMMs which is due to the fact that every frame comes from an independent "throw of dice" (see Section 7.2). The reasonable way to surpass this limitation is to jointly produce all the frames of a segment, as described in Chapter 7.

overcome the frame independence assumption

The main problem when assigning a fixed dimensional feature to each segment no matter its length is that it is not possible to compare the likelihoods computed from different segmentations even if they are associated to the same fragment of the observed signal since the set of features is associated to *different observation spaces*,[85] as illustrated in Figure 48.

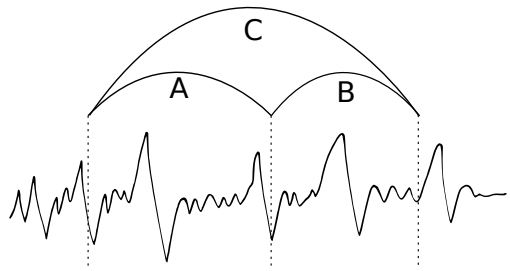


Figure 48: Fixed dimension feature segments lead to different observation spaces: segment C is associated a $\mathcal{R}^n$ vector but, when the same signal is segmented into A and B, it is represented by $\mathcal{R}^{2n}$ parameters.

Note that this problem is not inherent to the idea of representing each segment by means of a fixed dimensional feature vector: this problem does not exist when only one segmentation is taken into account. In this way, some SMs systems avoid this problem by using a

85 A similar problem appears when using ambiguous front-ends, see Section 2.9.2.

unique segmentation[86] obtained by a previous recognition algorithm, which is an obvious limitation. A more recent example is the system for isolated offline handwriting recognition described in [Bianne-Bernard *et al.* 2011; Section 5], where a segmentation process is applied to divide the handwritten word into a linear sequence of pieces. This makes segmentation mistakes unrecoverable. Despite that, the consequences of this problem are reduced in this last example because the segmentation is not performed at the character level. Indeed, segments are not necessarily associated to an entire character but possibly to a sub-part of a character which they misleadingly call *"grapheme"*.[87] Also, each character is modeled by means of a possibly variable number of parts using a HMM with Bakis topology (see Section 7.3.1). This recognition system is based on the use of MLPs to estimate the posterior probability of a fixed set of features given a variable length segment.[88] Unfortunately, not explanation is given relating how these posteriors are used on recognition. Approaches based on obtaining a sole segmentation are also related to non-fixed (or variable) frame rate parametrization.[89]

The problem of comparing hypotheses from different observation spaces cannot be avoided in general, since it is not reasonable to assume that we can obtain "the correct" segmentation.[90] This problem has been either ignored or approached by means of heuristics (e.g. by including some type of penalization to compensate the bias of these systems towards hypotheses using few long segments).

In order to deal with the problem of different observation spaces, [Glass *et al.* 1996; Glass 2003] propose to take into account the *entire observation space*, i.e. the set of feature vectors of all segments from all possible segmentations including not only those who correspond to the *correct* segmentation but also others who cannot be associated to any valid segmentation. The elaboration described below is based on the assumption that the content of incorrect segments can be properly characterized by means of an *anti-model*. We believe that this assumption has several shortcomings since an incorrect segment, taken isolatedly, may be similar to a correct one, as illustrated in Figure 49, whereas in [Glass 2003] incorrect segments are identified with segments which are not correct units.[91]

modeling negative examples

86 The qualifier "unique" has been included to emphasize that just one path is considered. Unfortunately, some authors use the term "segmentation" to talk about what we have coined "oversegmentation" which explains our concerns about ambiguities.

87 Clearly, this term is not used in [Bianne-Bernard *et al.* 2011] in a standard way.

88 A feature extraction process is applied to the bounding box of every segment in order to obtain 74 geometrical features which are fed to a MLP, also provided with a softmax output layer, in order to estimate the posterior probability of each class.

89 The idea of non-fixed frame rates has not to be confused with other techniques such as frame-skipping where some frames are ignored in order to reduce the computational cost, as proposed in [Woszczyna 1998; Section 5.3]. Despite the resulting frames being not produced at a fixed rate, each frame is obtained from a similar windowing of the speech signal (in ASR) so that any frame can take profit of the same type of state conditional likelihood estimation models.

90 Let us remember the Sayre's paradox, not to mention that the idea of just one correct segmentation can be questioned.

91 Note that [Glass 2003] is limited to ASR whereas we are taking a more general case and this fact has to be taken into account. Nevertheless, we do not agree on this assumption for ASR either since, from our point of view, some incorrect speech segments (e.g. a fragment of a vowel) are similar to correct ones.

Figure 49: An incorrect segment has the same shape as another well formed letter. In this case, a fragment of "m" resembles an "n". Also, left part of a "d" usually resembles a "c" and so on.

Decoding tries to find the word sequence W maximizing $p(W|A)$, where A is the set of acoustic observations associated with the speech waveform:

$$W^\star = \underset{W}{\operatorname{argmax}} \sum_{\forall S,U} p(S,U,W|A) \approxeq \underset{S,U,W}{\operatorname{argmax}}\, p(S,U,W|A)$$

The last approximation is based on the assumption that only one segmentation is correct. This segmentation is associated with a unit sequence U which corresponds to the recognized words[92] W. The expression $p(S,U,W|A)$ can be decomposed as follows by means of Bayes theorem:

$$p(S,U,W|A) = \frac{p(A|S,U,W)\, p(S|U,W) p(U|W) p(W)}{p(A)}$$

where:

- $p(A|S,U,W)$ is usually known as acoustic modeling.[93] It is also assumed that acoustics only depend on the unit sequence: $p(A|S,U,W) = p(A|S,U)$;
- $p(S|U,W)$ is the probability of the segmentation itself, which is obtained prior to A. Note that this corresponds to the use of explicit duration models (see Chapter 7);
- $p(U|W)$ is the pronunciation modeling;
- $p(W)$ is the language model (see Chapter 6).[94]

As explained before, the total observation space A is composed of both correct segments, X, and incorrect ones, Y. Note that, in this way, $p(A|S,U) = p(X,Y|U)$ where S can be removed because it is implied by X.

The following approximation can be obtained with the strong assumption of independence between X and Y, given U, and by observing that $P(X,Y|\text{anti-model})$ is a constant:

$$p(X,Y|U) \propto \frac{p(X|U)}{p(X|\text{anti-model})}$$

92 This additional level or stage where words are composed by units can be taken into account in the TSGM, but we have avoided to distinguish it explicitly, in the general description, because it is task dependent.

93 We have maintained the original nomenclature "acoustic modeling" of [Glass 2003] although the same formalism can be considered for other joint segmentation and classification tasks.

94 $p(U|W)p(W)$ roughly corresponds to the first stage of the two stage generative model, although the inclusion of pronunciation modeling in the second stage can also be envisaged when words, instead of sub-word units, are interpreted as units of the hidden sequence. Again, this is somewhat task dependent since other tasks might not require a lexicon.

Further assumptions relating the independence between the realization of different segments leads to this final equation where n is the number of segments:

$$W^\star \approxeq \underset{S,U,W}{\operatorname{argmax}} \prod_{i=1}^{n} \frac{p(x_i|u_i)}{p(x_i|\text{anti-model})} p(s_i|u_i) p(U|W) p(W)$$

The use of a sole model to deal with incorrect segments is a quite strong simplifying assumption. Unfortunately, the use of multiple classes is not compatible with the trick used to avoid taking into account all segment observations. A refinement of this approach called *near miss* modeling [Jane 1998] considers that incorrect segments are near misses[95] of a correct one. An effective decoding strategy has been proposed based on the assumption that incorrect segments can be partitioned and associated to correct ones.

mixed segment and landmark modeling

It is not very difficult to include a particular type of landmarks in these approaches. As described in Section 2.9.1, landmarks are often related to a set of features extracted at segments boundaries but they are not necessarily restricted to them. Nevertheless, some authors identify landmarks and boundary segment information and, hence, [Glass 2003] explains how to deal with this type of landmarks (quite similar to variable-frame-rate systems) by including the set of *all* landmark feature vectors Z (being associated to the correct segmentation S or not) in the joint space A which now becomes XYZ. Finally, further assumptions on the independence between Z and XY allows the inclusion of these features into a decoder.

There are other segment based approaches in the literature based on extracting fixed length feature vectors form variable length segments. The direct approach described in [Tóth 2006; Chapter 6], not very different from [Glass 2003], is based on the following decomposition:

$$p(W|X, L) \approxeq p(W|X) p(W|L)$$

where L represents the language model. The segmentation is introduced as a latent variable to be marginalized:

$$p(W|X) = \sum_{\forall S} p(W, S|X) = \sum_{\forall S} p(W|S, X) p(S|X)$$

The author acknowledges that approximating $p(S|X)$ with

$$p(S|X) \approxeq \prod_i p(s_i|x_i)$$

does not lead to a proper and normalized distribution, so the following alternative approximation is proposed:

$$p(S|X) \approxeq \prod_i p(s_i|x_i) \prod_{s \in \bar{S}} (1 - p(s|X(s)))$$

being $\bar{S}$ the set of segmentations other than S. The contribution of these segments is based on a model constructed with anti-phone examples. For performance reasons, $\bar{S}$ is limited to those segments which are

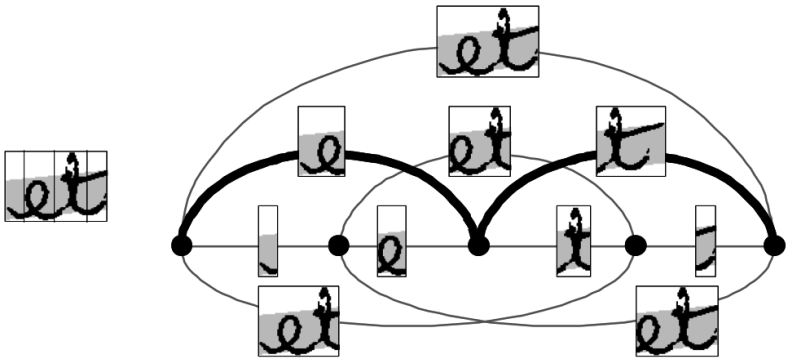


Figure 50: Figure taken from [Tay *et al.* 2001; Fig 1] illustrating an over-segmented word on the left and the corresponding segmentation graph on the right. Each edge corresponds to a character hypothesis. Such a segmentation graph is similar to the partial lattice graphs of [LeCun *et al.* 1998].

near-misses of the correct ones. This work is based on artificial neural networks to construct the different classifiers.

In more recent works based on fixed dimension feature vectors to model entire segments, the problem of comparing different observation spaces is simply ignored. [Tay *et al.* 2001] addresses an offline isolated handwriting task by dividing handwritten words into slices by means of an over-segmenter based on geometrical heuristics.[96] A segmentation graph is obtained by joining consecutive slices afterwards, as depicted in Figure 50. These segments are considered as candidate characters. A fixed dimension feature vector composed of geometrical features is obtained from each segment and is fed to a multilayer perceptron with a softmax output in order to estimate the conditional probability of each character class given the segment. In order to cope with segments which may possibly not correspond to any character, an additional output estimates the probability of being feed with a wrong[97] segment, but it is not trained (even with wrong patterns). The main purpose of this extra class is to absorb some probability mass, hence avoiding the sum of all other classes to be one. Instead of posteriors, outputs are used as pseudo-likelihoods without dividing by priors, although no satisfactory explanation or theoretical justification is provided.[98] These values are used as emissions in a HMM, although it is also questionable to call them "HMM" when all transition probabilities are set to 1 (and never trained) and when individual emissions may correspond to segments of different length.

95 That is, segments which are nearly overlapped with a correct one, which seems to us quite difficult to quantify.

96 The over-segmenter chooses points minimizing the number of ink pixels

97 Which is called "junk class" by the authors, not to confuse with the garbage symbol used in HTR or the noisy event modeled in ASR. In [LeCun *et al.* 1998] the term "rubbish" is used instead. The absence of nomenclature uniformity is widespread.

98 The MLP is trained by means of gradient descent with a global level criterion, but this does not suffice to make the method mathematically sound, at least as it is described in terms of probabilities. This approach also resembles to us the "graph transformer networks" proposed by [LeCun *et al.* 1998; Bottou and LeCun 2005] described in Section 4.4.4, although they *sagely* avoid the use of terms from probability theory and describe their system in terms of energy based models.

Figure 51: Example using a handwritten word from iam database to illustrate some drawbacks of using frame-wise posteriors for decoding: The image corresponds to the word "but", but a nonzero probability is also assigned to "lrit" sequence, which is perfectly reasonable, but also to sequences such as "lbruit" which does not correspond to a valid segmentation.

We are very critical with the use of segment models based on extracting a fixed length feature vector from each possible segment. The problem of comparing different observation spaces is either ignored or tackled by adopting very strong independence assumptions as well as other unjustified heuristics. The lack of proper (probabilistic) justifications explains why some authors using these ideas describe their systems in terms of scores.

We have to remark that segment models of generative nature, explained in detail in Chapter 7, do not suffer from this problem. On the other side, the use of a finite size feature vector to represent a variable length sequence is perfectly fine with discriminative models: Indeed, if a segment x is represented by means of a fixed dimension representation $f(x) \in \mathbb{R}^n$ and we estimate $p(u|x)$ by means of $p(u|f(x))$, there is not much problem. But when $p(x|u)$ or $p(x, u)$ have to be estimated, we must not forget that this is not the same as estimating $p(f(x)|u)$ or $p(f(x), u)$, respectively.[99] That is why we restrict the use of this type of discriminative segment models, in our work, to re-score frame-based generative models, as proposed in Section 7.7.

4.4.3 Direct estimation of frame-wise segment posteriors

This section addresses a particular decoding approach based on estimating the conditional probability of observing each type of hidden unit at every position of the observed signal. Let us first make some remarks in order to better describe this approach:

- the estimation of frame posteriors is a problem by itself and can be formulated in a sound way for many types of models. For instance, it can be computed for TSGMs by means of marginalization over the posteriors obtained at the segment level (more detail in Section 4.3). Remark that frame posteriors contain less information than a segmentation and classification task and, therefore, some information is lost in the process. The approach described in this section can be considered, in some way, as the reverse problem: retrieve the sequence of hidden units given the frame posteriors. Since segmentation constraints are not taken into account when retrieving frame posteriors, their use in decoding may pose problems as those illustrated in Figure 51;

- as probably observed by some readers, frame posteriors are also used to obtain scaled likelihoods to model the emission probabilities in HMMs: They are converted into scaled likelihoods by means of Bayes theorem in the HMM approach, which is not the

Hybrid HMMs are different

99 Indeed, $p(f(x)|u)$ would be related to $\sum_{y \in f^{-1}(f(x))} p(y|u)$.

case here. This particular type of HMMs are usually known as "hybrid" HMMs, see Section 7.5.3 or [Bourlard and Morgan 1994]. Nevertheless, the approach described in this section is different because these posteriors are used in a different way, and also because they are necessarily associated to hidden units whereas in the HMM approach they are associated to types of emission probabilities;[100]

- the fact that this approach is discriminative at the frame or local stage is not the same as being discriminative at the global (or sequence) level. Indeed, excepting some approaches which obtain the frame posteriors from entire sequences (as is the case of RNNs) and are capable of dealing with language model information, these posteriors are just suboptimal approximations of the second stage of TSGMs (they are like acoustic models in ASR) which are not able to deal, in general, with Sayre's paradox;

- there may exist different ways to obtain frame posteriors but, once these posteriors have been obtained, similar techniques can be applied to obtain the sequence of hidden labels afterwards.

Two particular ways to obtain frame posteriors will be examined: sliding windows and recurrent neural networks.

Overlapping sliding segment recognizer

The use of sliding windows to obtain a sequence of feature observations or frames seems the most natural method to obtain discrete sequences from continuous signals.[101] One of the particularities of sliding windows is that they are often partially overlapped since the window length should be wide enough to obtain local features (e.g. some time-frequency features in ASR or some contours in HTR) and, at the same time, the frame rate should be high enough to capture the dynamics of the system.[102] Despite that, features associated to a sole frame may be insufficient to estimate which phoneme or grapheme is being pronounced or written at a given position and all the frames comprised in a given segment are rather required to perform this task.

In an overlapping sliding segment recognizer, the sliding window traversing the frame sequence is wide enough to capture an entire segment in the hope that it is possible to detect each possible type of hidden unit and to classify with this information the centering frame,[103] as illustrated in Figure 52.

100 In this way, some type of emissions may be shared by several units and a sole unit may use several types of emissions. We have to admit that some hybrid HMM systems use a sole type of unit for each type of hidden model, this is only a particular case which, unfortunately, is used without performing more exhaustive trials on the number of states, even in quite recent works as in [Plahl 2014].

101 Although it can also be applied to discrete sequences

102 That is, it should have a resolution high enough to place different units at different frame positions

103 Not necessarily the centering frame, it is possible in general to talk about the left and right contexts which usually have the same length. The sliding window used for this purpose can be made of frames obtained from a previous feature extraction stage also based on sliding windows.

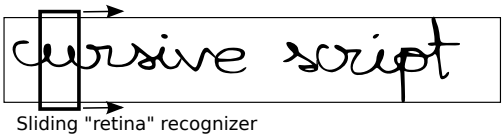


Figure 52: Figure showing how a sliding recognizer can be swept along the sequence to check each position in the sequence to perform, in this case, "character spotting". In the example, the recognizer is over the "u" character, but the retina is usually faced to incorrect segments or even to segments longer than the window width (other figures can be found in the literature illustrating these ideas: [LeCun *et al.* 1998; Fig. 22], [Lee and Kim 1999; Fig. 4], or [Rashid *et al.* 2012; Fig. 4]).

A discriminative classifier is hence required to compute the posteriors; for instance, in the area of HTR, [Rashid *et al.* 2012] proposes the use of a fixed sized sliding window used to feed a multilayer perceptron (MLP) trained to detect graphemes and to produce a sequence of posterior probabilities. There are other works in the literature producing a vector of frame scores with connectionist models, namely: convolutional networks describedin Section 4.4.4 and recurrent neural networks described in the following subsection.

Another assumptions are required in the sliding window approach besides the requirement of a maximum window length wide enough to capture an entire segment:

- the classifier has to learn to ignore the non relevant parts of nearby segments inside the window when the segment length is lower than the window width; and
- it has to detect incorrect segments as well.

adaptable window width

There exists other works based on sliding windows which try to adapt the window width to the expected width of the segment. A remarkable example is proposed in [Fischer 2012] where a set of graphs from the input sequence using several window widths are obtained as illustrated in Figure 53. In order to convert this quite unusual representation into a sequence of feature vectors, a set of distances[104] is computed by means of a graph embedding: graphs are compared with a set of prototypes, the width of the sliding window depending on the width of the prototype.

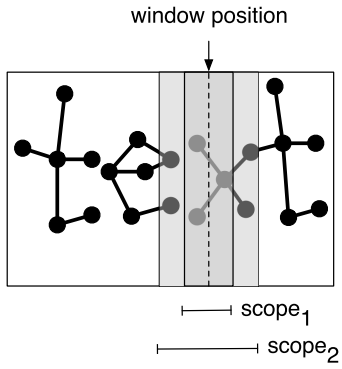


Figure 53: (taken from [Fischer 2012; Fig 4.6]) illustrating how a variable width sliding window is used to obtain sub-graphs from a graph sequence in order to apply the dissimilarity space embedding approach to get "Graph Similarity Measures".

104 Distances are normalized as described in [Fischer 2012; Section 4.5.1] but they cannot be considered posteriors but rather scores.

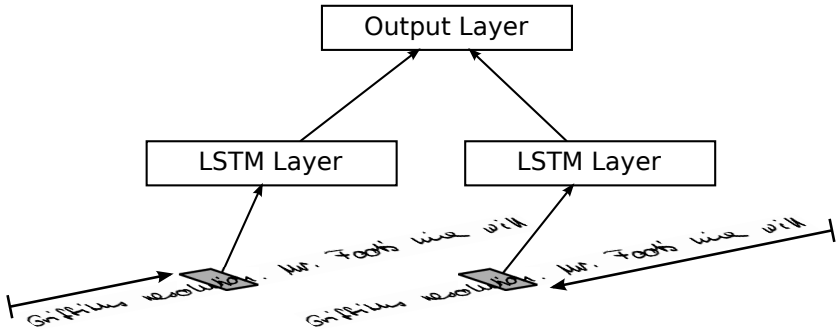


Figure 54: Figure taken from [Frinken *et al.* 2012b] illustrating how a BLSTM RNN works.

RNNs with connectionist temporal classification (CTC) layer

One of the limitations of the sliding window approach is the window width which limits the maximum segment length the system is able to recognize. Recurrent connectionist models do not suffer from this problem since the entire signal is fed to them in a frame-by-frame basis. Moreover, they have the following advantages:

- its internal state makes it possible to span the influence of past observations over the entire sequence prefix, which qualifies them to model not only the way a hidden unit is associated to an observed segment but also to model which sequences of hidden units are more likely (what is known as language modeling). On the other side, the decisions made in the sliding window approach are taken independently and their capability to model unit sequences comes from the use of a wide overlapped window which may contain past segments in some cases;
- the efficiency is not degraded when considering wider and wider windows as is the case of the sliding window approach;
- the fact of feeding a maximum length sequence in the sliding window approach forces the classifier to ignore the context of narrower segments, which are part of a large[105] input, but this is not the case in the RNN setting where the segment to be classified is tackled by the internal state.

Since frames are fed frame-by-frame, the network does not have the right context of the last frame at its disposal. The dependence on right/future observations may be addressed in several ways: by delaying the output a number of frames associated to the right context, by using an explicit right context (which increases the input size) or by using a bidirectional recurrent network as proposed in [Graves and Schmidhuber 2005] (depicted in Figure 54). This last work also has two remarkable particularities: on the one side, the use of long-short term memories (LSTMs) makes them robust against the "vanishing gradient" problem (the exponential decay of activations with the increase in the number of layers or cycles); on the other side, a special "Connectionist Temporal Classification" (CTC) output layer is based on the

105 Remember the "curse of dimensionality".

softmax output (so that RNN outputs can be used as probabilities) which contains an additional output of "observing no label" (useful for decoding, as described below). The main advantage of the CTC layer is its capability of using information from the decoding stage in a gradient descent MAP training of the entire RNN which does not require the observed sequence to be segmented.

Decoding with frame-wise posteriors/scores

Relating the use of a vector of frame posteriors, the most common way to obtain the final sequence of hidden labels from it can be described as a many-to-one map which removes repeated labels as described in [Graves *et al.* 2009]. Let us observe that this technique is not useful, in principle, for tasks where the location of labels must be exactly determined. In order to cope with repeated labels, an additional unit type is included to estimate the probability of observing any label (represented by a special "blank" symbol). A sequence containing the same label repeated n times can be represented provided there is at least $2n-1$ frames in order to assign at least one of them the correct label and blanks to separate them.[106] The frame to label sequence mapping removes first repeated labels and then the blanks, but this process is not as simple as choosing the most probable class at each position since additional constraints are usually required. In this way, the removal of repeated labels and blanks can be considered just as a "compression" of the feature vector sequence,[107] whereas the choice of a particular label sequence can be efficiently computed over all possible combinations with a dynamic-programming algorithm which takes both a lexicon (to force words to correspond to allowed valid sequences) as well as a language model into account. Indeed, this algorithm [Fischer 2012; Section 6.3] is nearly identical to one of the most successful and widespread approaches to decode HMMs (to be described later in this work, mainly in Chapters 10 and 12), namely: the combination of token passing, prefix tree organization of dictionary words, and pruning techniques to obtain an output lattice.

many-to-one map

Although frame-wise scores admit a probabilistic interpretation (they can be considered posteriors in most cases), the decoding process just described does not have a proper probabilistic interpretation.[108] Nevertheless, this approach has been empirically validated yielding very impressive results in many tasks including offline recognition of handwritten lines (HTR) in an unconstrained domainand phoneme recognition, to mention a few. This approach has also the curious particularity of being one of the very few successful "fully connectionist" approaches to these tasks.[109]

106 Which reminds us of the BIO tagging ideas described in Section 2.3.

107 As described in [Fischer 2012; Section 6.3], who achieves compression factors between 6 and 10 for the data sets considered in his thesis.

108 Indeed, many works adopting this approach produce just a sequence of scores. In this way, some probabilities can be considered automatically *promoted* into scores when a probabilistic interpretation is no longer valid and, conversely, some models using scores may have a proper correspondence with a probabilistic model in other cases.

109 Meaning that other connectionist approaches such as, for instance, a decoder using Hybrid HMM/ANN acoustic models and Neural Network LMs are nearly but not fully connectionist. More recently, works with convolutional and deep neural networks are also giving very promising results.

A drawback of this model is the requirement to learn the sequence of units (lexicon and LM) from the observed sequences (which does not have to be explicitly segmented). Other models seem more orthogonal regarding this issue since they allow us to train the lexicon and LM from a large textual corpus, making it easier to adapt the system to different tasks.[110]

The main critique of using frame posteriors, when considering the applicability of this technique to the general problem of the joint segmentation and classification of sequences, is that it ignores the Sayre's paradox by the inherent limitation of the representation (see, again, Figure 51). This approach does not admit a generative interpretation since the evidence is not associated to the entire observation space (e.g. different sequences may correspond to different number of frames depending on the compression stage) and some frames may have been ignored or taken into account repeatedly during the process depending on the choice of hidden units.

see also hybrid and tandem models

Let us also remember that the same framewise values, either posteriors or just scores, can be used in other ways: to estimate scaled likelihoods or as additional features to be concatenated to the frame sequence (see *tandem* models in Section 7.5.3).

4.4.4 Graph transformer networks (example of an energy based system)

Graph transformer networks (GTNs), as described in [Bottou *et al.* 1997; LeCun *et al.* 1998; Bottou and LeCun 2005], are claimed to be an unified design paradigm to construct recognition systems that:

- integrates several modules in such a way that they can be trained by means of gradient descent techniques with a global performance criterion;
- modules used to construct GTNs may be of heterogeneous nature but can be used in the same gradient descent learning method provided that the function they implement is continuous and differentiable (almost everywhere[111]) with respect to its inputs and its internal parameters;
- are not based on probabilities: they are an example of energy based models which, in turn, can be described as graphical models (Section 4.1.1) where each configuration of the variables is associated a scalar value or "energy" measuring the degree of compatibility of the configuration (the lower the energy the better).[112] In this way, what we seek is to find the value Y which minimizes $E(X, Y)$ instead of the one which maximizes $P(Y \mid X)$. A possible drawback of this approach is that values from different sub-systems are scaled differently and, hence, difficult to combine;

110 The acoustic models in some ASR systems can be trained with a corpus which is only related to the task at hand by acoustic criteria.

111 There may exist some special modules (e.g. the min function), which are non-differentiable but remain differentiable "almost everywhere".

112 This, supposedly, solves the label bias problem associated to Maximum Entropy Markov Models where some observations can be ignored by the model, see [Lafferty *et al.* 2001].

- the type of values received or generated by some modules are not restricted to numeric vectors: directed acyclic graphs can be used to represent sets of weighted sequences. Some modules able to receive a graph and to produce another one are called "graph transformers" although, in most cases, these modules seem to apply just the same operation over all the edges of the graph.

Let us briefly discuss which different module types are relevant to solve the joint segmentation and classification problem and how they can be combined to construct GTNs for that purpose. The resulting systems can follow an explicit[113] or an implicit segmentation approach.

In the first case, a segmentation graph is produced by associating graph vertices to possible cutting points obtained in a previous oversegmentation, whereas the segments comprised between the corresponding cuts constitute the graph edges.

When using the external segmenter approach, a special "recognition transformer" module receives the segmentation graph and produces a score associated to each segment which measures how likely this segment is related to each class (an additional penalty given by the segmenter is also taken into account). The output of the recognition transformer is an "interpretation graph" which reminds us of the graph based approach described in Section 4.4.2 excepting the fact that, here, scores do not have a probabilistic interpretation. Some works proposing a similar procedure transform the image associated to the segment into a fixed length feature vector although, in this case, a convolutional neural network (CNNs) are used instead. CNNs are based on the replicationof the same neurons along the sequence (over the time axis, in the case of time series).

convolutional neural networks

The interpretation graph may be fed to a module known as "grammar transformer" which combines the graph with a grammar using a transducer/automata composition algorithm (see Chapter 5). The final step is a "Viterbi transformer" which simply extracts the best path from the directed acyclic graph received at the input by means of dynamic programming.

The second approach, which does not require an explicit segmenter, is based on sweeping a segment recognizer along the sequence. This recognizer can be implemented with a neural network centered at the segment to be detected or by means of CNNs. A vector with scores measuring the presence of each class at each input location, similar to the previous sliding window approach, is produced. An interpretation graph is obtained from this vector by composing it (considered as a linear graph) with a character model transducer.[114] Once the interpretation graph is obtained, the rest of the architecture is similar to the previous case based on the segmenter graph.

The parameters of GTNs modules can be trained using a global criterion. To this end, some additional modules, not required after training, can be added to the basic recognition architecture in order to employ one of the following estimation methods:

113 The "external" segmentation approach is also known as HOS or INSEG in the literature.

114 Although the use vector of frame-wise scores is similar to the approach described in Section 4.4.3 and depicted in Figure 52, the use of these values would make it more related to hybrid HMM systems when ignoring some probabilistic interpretation issues.

VITERBI TRAINING the correct label sequence is used, during training, to constrain the interpretation graph (in order to make all paths compatible with the correct label sequence) before being used by the Viterbi transformer. The result is sent to another module known as "path scorer" which computes the penalty associated to this sequence and which is backpropagatedthrough the system in a reversed way.

DISCRIMINATIVE VITERBI TRAINING corresponds to the discriminative version of the Viterbi training. Discriminative techniques were briefly discussed in Section 4.1 and basically consists in not only trying to minimize the penalty associated to the best path but also to increase the penalties of incorrect ones. This training technique can be implemented by means of the same Viterbi transformer module which is applied, in this case, to the constrained interpretation graph and to the original interpretation graph as well in order to compare both results.

FORWARD TRAINING It is widely accepted that the Viterbi procedure is just an approach since only the best path (e.g. only one labeled segmentation) gives support to the evidence whereas there may exist many other hypotheses which may provide some contributions. The forward training case is similar to "Viterbi training", but a "forward scorer" module replaces the "Viterbi transformer" in order to obtain a forward penalty which takes into account not only the lowest penalty but other paths which can be associated to the right answer.

DISCRIMINATIVE FORWARD TRAINING can be viewed as a version of "discriminative Viterbi training" based on the "forward scorer" of the previous case. It is not clear for us how to use a forward scorer to efficiently search the best interpretation. This will be discussed in Chapter 5 and is related to [Casacuberta and de la Higuera 1999].

Although this approach seems quite different from the HMM mainstream, we believe that this is mostly due to the use of a different nomenclature and the idea of using a global gradient descent training technique. If we ignore the fact HMMs use a probabilistic setting with scaled likelihoods, they are not so different.[115] We can also find a similitude between this approach and the one we have adopted in our work, as the reader will observe. In both cases, a system is constructed from simpler pieces taken from a finite repertoire.[116] Our system is described in terms of a dataflow architecture (described in Chapter 8) where graphs can be sent from one module to another, which is quite similar to a graph composed of modules although, in our case, graphs are not sent atomically from one module to another but are rather serialized or divided into little pieces that some algorithms are able to use directly instead of waiting for the entire graph to be received.[117]

115 Indeed, as stated in [LeCun *et al.* 1998]: *"Unfolding an HMM in time yields a graph that is very similar to our interpretation graph"*.

116 The use of dataflow approaches in pattern recognition is not limited to GTNs and ours.

117 We do not know the implementation details of GTNs relating how graphs are sent between modules, but we have not found any of such claims in the literature.

4.4.5 Some non-probabilistic frameworks

This final section sketches some non-probabilistic approaches which are mainly based on distances and on discriminative classifiers. They may overlap with models discussed before and can also be related to the probabilistic setting (it is possible to estimate probabilities from distances or scores in several ways, not to mention non-parametric kernel density estimation techniques). Many of them are only suitable for classifying a signal into a discrete and finite set of labels, which is a severe limitation w.r.t. other models. These methods can be relegated to segment classification (e.g. isolated word recognition instead of the continuous counterpart) or as pieces of the second stage of the TSGM (e.g. segment estimators, described into more detail in Chapter 10).

Distance-based systems

They are usually related to template based systems and to non-parametric models.[118] These models are also related to holistic approaches and, in general, do not impose independence assumptions among nearby frames as is the case of HMMs. Despite these advantages, they often model entire words instead of sub-word units and suffer from the curse of dimensionality as mentioned before when dealing with the segmental approach of Section 4.4.2.

It is not clear how these approaches could be extended to cases where the sequence of hidden labels is not finite. The obvious solution is to use these models at the second stage of a TSGM using words instead of letters as hidden labels. This idea was indeed the non-probabilistic precursor of TSGMs described in this work, see for instance, [Sakoe 1979; Hiroaki 1982]. An speed-up can be obtained in these systems by means of fast nearest neighbor search techniques [Aibar 1997]. More recent works integrated these distances with language modeling [De Wachter *et al.* 2007] whereas others used templates to re-rank a list of N-best hypotheses obtained in a previous HMM system [Aradilla *et al.* 2005].

Code-breaking

The re-ranking or re-scoring process approach of [Aradilla *et al.* 2005] described before has some resemblances with a curious way to allow the use of discriminative models in the joint segmentation and classification of segments known as acoustic *code breaking*[119] It was first proposed by Jelinek and later studied, among others, in [Venkataramani 2005] or in [Ragni 2013; Chap. 4]. It is based on the divide-and-conquer strategy since the recognition of the entire signal is divided into multiple segment classification problems. The usual procedure is as follows:

divide and conquer

118 The model for a given class is a set of exemplar templates. Training basically consists in selecting these subsets.

119 The name come from the fact that, as with code–breaking, we are in an scenario where we are willing to devote an indefinite amount effort to decode a given utterance. Indeed, Jelinek points out that "everything would be allowed excepting human intervention". This remembers the curious fact that some approaches surpass this limit, let us remember interactive systems and the use of *captchas* described in Section 3.2.2.

1. perform an initial recognition step with a recognition system whatsoever (usually, but not necessarily, based on HMMs). Some works take into account the word segmentation of the best hypotheses [Gales and Flego 2010], others are rather based on systems able to produce a word graph or confusion network representing the best hypotheses [Layton 2006];
2. determine the problematic and confusing regions;
3. resolve these problems by means of models specialized on these particular confusions. These models can be discriminative classifiers which can be determined and trained beforehand from a list of frequent confusions encountered in a training dataset (e.g. binary classifiers trained on confusable pairs).

Although this a set of local decisions may empirically improve the WER, it seems to lack a global criterion related with the LM. An alternative way to use discriminative models at the segment level in a generative setting is proposed in Section 7.7.

Representation learning

As we have observed, many proposals are based on Dynamic Time Warping (DTW) to estimate the distance between variable length sequences, but there are other successful alternatives based on first mapping these sequences into a fixed dimension space. Indeed, some techniques are more general because they are able to deal with patterns that have been never seen during training (a capability often coined as *zero-shot learning*) provided that the associated sequence of hidden units is provided (e.g. to recognize words from their transcription in a HTR task). A poor man's solution is the *recognition by retrieval framework* proposed in [Goel *et al.* 2013] for the task of text word recognition. In this approach, a subset of synthetic word images is constructed from the lexicon and the classification is performed by comparing the input with the elements of this subset. A more novel approach able to deal with words that have never appeared in the training set consists in learning the appropriate representations: a deep connectionist model is used in [Bengio and Heigold 2014] to classify speech segments into words in such a way that a mapping from a representation of the associated transcription[120] to the internal representation of the acoustics is also learned. This mapping would place nearby words which *sound alike*, allowing the use of nearest neighboring techniques even for words never seen during training. Similar ideas based on embedding both the observed signals and a representation of label sequences into a common subspace are also described in [Rodriguez-Serrano *et al.* 2014; Almazan *et al.* 2014]. In this case, the representation of label sequences is based on a *pyramidal histogram of characters* which is just the concatenation of bag of label features of different subdivisions of the transcription at different scales. In this work, the extraction of features from the observed signal (a word image) is also obtained from different subdivisions. A mapping of both spaces into a common subspace allows the meaningful measure of distances between signals and their possible transcriptions.

zero-shot learning

representation learning

120 Using a bag of n-grams, see Chapter 6.

4.5 SUMMARY AND SOME CONCLUSIONS

Previous chapters have motivated the interest in some particular kind of problems and problem types. Now, a model family suitable for dealing with the joint segmentation and classification on sequences has been proposed: two stage generative models (TSGMs). The following chapters will be devoted to study their different parts and also to review and propose some algorihtms and their implementations.

TSGMs are described

Two stage generative models are very related to segmental models (SMs) and with two-stage recognition systems. Historically, the term "two-stage" or "two-pass" has been almost restricted to distance based template systems (e.g. based on DTW). But, soon, HMMs became the dominating model type for decades[121] and the most reasonable way to use them is by means of one-pass decoders. Since HMMs eclipsed other alternatives, the distinction between two-stage and one-pass became basically unnecessary.

HMMs have dominated for decades

Although these models are not new, they are unusual in the sense that other works that can be found in the literature do not describe them as is although, in the best case, they describe related albeit different model types. In this way, most works are exclusively focused on HMMs. Relating models other than HMMs, there is a minority of works discussing the more extensive family of SMs whereas much others prefer to deal with graphical models (GMs).

In this work, we pretend to reclaim the term two-stage in order to subsume what has been previously seen a plethora of ad hoc models and algorithms mainly related to SMs and to other extensions of decoding. They are also related to time-conditioned decoders [Ortmanns and Ney 2000] (see Chapters 5, 10 and 11) and, to a certain extent, to multipass approaches where a graph is used as an intermediate representation, although the graphs used in multipass systems may differ from the DAG produced in the first stage of two-stage decoders.

Our aim for a general and unified view of decoding probably points in the same direction and has the same spirit of early works such as [Godin and Lockwood 1989] or [Aibar *et al.* 1992] in the sense that a hierarchy of problem types based on more or less strict assumptions is taken into account and is associated to their corresponding algorithms solutions. In this context, a hierarchy for the second stage has been proposed. The quest for a unifying point of view is not novel since other works have already generalized decoding and models in terms of semirings, weighted logic or hypergraphs. Indeed, we will say more about them in the next chapter.

proposal of a TSGM hierarchy

Meanwhile, one of the most successful, promising and general frameworks are GMs. Dynamic GMs seem quite well suited for the problem type we are dealing with. Indeed, and in order to better describe and to put TSGMs in context, this chapter has introduced a short review of modeling techniques from the machine learning and statistical pattern recognition points of view, paying special attention to carefully review probabilistic GMs. Although these reviews could have probably been replaced by a *well chosen* set of bibliographical references,

121 Recent works on end-to-end approaches based on connectionist deep models seem the current trend.

synthesis effort

no individual choice would *exactly* fit or match our goals. This synthesis effort has produced what, from our humble point of view, is an unconventional and unifying point of view of GM algorithms: unconventional because it is mainly focused on factor graphs (FGs), and unifying because the connection between variable elimination, jointree and summary-product (a.k.a. belief propagation) algorithms has been clearly established, which is not common in the outstanding amount of literature on this matter. A notable exception is [Darwiche 2009] where we can also find this relationship, although the exposition is focused on Bayesian networks instead of FGs. A more recent presentation of these ideas for FGs can also be found in [Sümer *et al.* 2011]. We hope that this perspective and choice of presentation will help readers to gain insight into the relationship of these algorithms which have been reinvented and proposed several times in quite different fields.

Another synthesis effort has been done in order to provide a review of alternative frameworks and models. Although it seems impossible to provide a complete review, this part is not exactly equivalent to any well chosen set of bibliographical references either. We expect that this comparative review will also be useful for the interested reader.

It is necessary to acknowledge the limitations of modeling and to distinguish between description and modeling. There may be problems that can be effectively tackled with TSGMs while, at the same time, they certainly admit more natural and proper ways to describe them. Relating the limits of TSGMs, we are not dealing with problems involving reordering or non local interactions.[122]

too broad family

Relating the three classical problems, TSGMs constitute a too broad family since they are basically specified by a restriction. It is required to assume additional constraints to obtain useful algorithms, as briefly outlined in Section 4.3 and further developed in the following chapters (specially in the next one). Different sets of independence assumptions in the first and in the second stage of TSGMs, as well as on the loss function used during decoding, would lead to different algorithmic possibilities and we would like to cast light on these relationships. An special interest has been put in this work in order to establish the relationship between two-stage and one-pass decoders. Furthermore, the relationship between regular and context free model parsing/decoding algorithms will be addressed in the next chapter.

are TSGMs useful or promising?

Relating the apparent supremacy of HMMs, some readers may observe that, historically, SMs have put the focus on their capability to overcome some HMM limitations. However, despite these claims, HMMs continued to be the preponderant choice. In this regard, it is justifiable to cast doubt on the practical relevance of TSGMs *other than HMMs*. We expect to contribute to dispel these concerns through this work. There is an strong trend towards connectionist deep learning techniques nowadays. Some of them are naturally integrated into the HMM paradigm, others are known as end-to-end approaches. We can expect, in the middle of both extremes, a revival of TSGMs associated with the use of segment based connectionist models.

122 An example of a problem involving reordering is SMT, an example involving non local interactions is the case of biopolymers which are described by biological sequences but they adopt complex geometric shapes.

5 RECOGNITION, DECODING, PARSING, TRANSLATION

I suppose it is tempting,
if the only tool you have is a hammer,
to treat everything as if it were a nail.

Mark Twain

CONTENTS

5.1 INTRODUCTION

Parsing is a huge, even intimidating, area of research with lots of formalisms, concepts and algorithms. This work is focused on some particular cases of statistical pattern recognition related to sequences where some kind of structure exists in both the hidden labels and the observed output. The most predominant examples discussed in this work (ASR and HTR) are related to natural language. Our approach to parsing is biased to this area and, not surprisingly, it is quite different from many classical works on parsing.

Most traditional parsing approaches, usually focused on compiler construction, are mainly addressed to the problem of parsing an unambiguous string with a restricted type of context free grammar (CFG) by means of an algorithm specially tuned for this purpose [Grune and Jacobs 2008]. This is not our case since we are rather dealing with many alternative hypotheses accompanied with probabilities.

our case

Let us remember that we have chosen to work with a particular family of generative models (described in Chapter 4) which we have called "two-stage" (TSGM) since a first stage generates a sequence of hidden labels and, later, each label generates/explains a segment of the observed signal. A general decoding approach for this model was described in Section 4.3.2. It operated the TSGM in a reversed way: a DAG was produced from the observed signal by using the segment models. This DAG is a compact representation of the set of possible segmentations of the signal where each arc is annotated with the likelihood that the associated segment corresponds to each type of hidden unit. The DAG is, therefore, a finite weighted language playing the same role as the input string of classical parsing descriptions.

parsing as intersection

The second part of the algorithm, more related to the current chapter, consists in combining this DAG with another weighted language which models the a priori probability of each sequence of hidden labels and which is known as *language model* (LM). Different types of information can be obtained from this combination (the most probable segmentation, etc.) that can be seen as the composition of weighted transducers. This approach constitutes a point of view closely related to the "parsing as intersection" (more generally, transducer composition) paradigm discussed below.

one pass algorithm

Obviously, both stages of the algorithm (generate an input DAG, combine it with the LM) can be interleaved and combined more efficiently than the previous naive description. For instance, some particular cases allow us to divide the problem into stages at different "cutting points", one of them being more efficient: the generation of segments using HMMs and the use of regular models to describe the language model. The most straightforward way to divide the process into two stages consists in dividing the signal into segments at the label level and to process them with HMMs, but it is also possible to integrate the HMM and the LM in the second stage and to restrict the first stage to the estimation of frame emission likelihoods. Since this approach avoids segmentation ambiguities in the first stage, which are divided into frames in a trivial way, the corresponding DAG is a linear sequence instead of the common set of spans more related to the transitive closure of a linear graph (which is quadratic with the sequence length). The second stage does not need to compute all the edges from the transitive closure. This algorithm known as *one pass* was mentioned in Chapter 4 and its linear cost will become more clear in this chapter and in following ones.

modeling up to context-free

Relating the formal models used to specify the LM, we will only consider the case of regular and context free languages. Their specific use for modeling LMs is delayed to Chapter 6 whereas the current chapter tries to provide some general background. In spite of the preponderance of regular models for language modeling, we believe that context free grammars constitute a good trade-off between representation capabilities and the disposal of parsing algorithms. That is why we will devote some effort to them, specially focusing on what some works call "generalized parsing" to denote the fact that they do not assume some special CFG sub-family.[1]

1 For example, LR(k), SLR, LL(k), etc.

The aim of this chapter is quite modest and threefold: 1) to provide some background to parsing and to formal languages, 2) to promote certain points of view which we believe deserve more attention than what is usually found in the literature and which are used, in this work, to describe some parsing algorithms, and finally 3) to show that it is possible to derive the aforementioned algorithms from the definitions of semiring operations on transducers and transducer composition.[2]

novel composition algorithm

As with other chapters, we have tried to organize a large amount of ideas in such a way that they seem a combination of simpler concepts, leaving aside many important topics of the area not required in a short review. Hopefully, this exposition can provide a simpler background for readers interested in this topic. The reason to invest some effort here is that we have not found this particular way of exposing the ideas in previous literature.[3] We do not hide our intention to boost or to privilege certain formalisms which are usually neglected in the current literature, specifically the use of recurrent transition networks (RTNs), which are formalized in this work in a different way, and the description of some related algorithms by using semirings without necessarily relying on hypergraphs [Klein and Manning 2001b] or on weighted logic deduction [Goodman 1999], as is usually the case.

shameless lobbying

Chapter organization

The rest of the Chapter is structured as follows: The following section will briefly discuss what is commonly understood by parsing and which are the differences among parsing, recognition and decoding, since those concepts are related and overlapped.

what is parsing?

The next section introduces the idea of weighted languages using semirings, the composition of weighted transducers and the extension of semirings to language and transducer operations are also explained. This section emphasizes the idea that weighted language and transducers are values which can be combined by means of operations and which constitute semirings themselves. Note that weighted languages and transducers are described here in an abstract way without resorting to actual models such as formal grammars or automata.

Another section reviews some formalisms to describe formal languages (formal grammars, finite state automata (FSA) and recurrent transition networks (RTN)) and the Chomsky–Schützenberger hierarchy. They are extended to the weighted case for context free languages and some sub-classes of CFGs and normal forms are briefly reviewed. They are useful to relate regular models with a particular normal form and its relationship with finite state automata and finite state transducers. The relationship between grammars and systems of equations makes more clear the interpretation of the graphical form describing those models, the sum of product associated to the spans of these graphs and the transitive closure of the underlying matrix.

2 Other works exist which first make similar definitions of weighted languages but, unfortunately, the description of parsing and composition algorithms is not related to them.

3 We can find some reasonable approximations [Ésik and Kuich 2007; Klein and Manning 2001b; Vidal *et al.* 2005a;b; Huang 2008b; Dyer 2010]. These references, and others, are detailed through the chapter.

Since we have limited our case of study to some types of models, we can restrict our attention to the composition of weighted transducers (which can be understood as weighted relations rather than specific models such as weighted finite state transducers) adapted to the case of the composition of a regular with a context free model.

A section is devoted to derive an algorithm from the interpretation of the graphical form of formal grammars, and from the definitions of transducer composition and the semiring operations. This algorithm is the flagship of some proposals described into more detail in Chapter 12. It has some resemblances with the well known Bar-Hillel algorithm (originally described for intersection of languages and later extended to semiring weights and top-down filtering [Dyer 2010]), with Earley parsing and with the composition of finite state transducers.

Let us observe that, until now, we have been mainly concerned with the computation of a particular semiring value. It is now the moment to relate that with other more classical parsing points of view more related to searching. We will see that certain specializations of the semiring computations can be considered when restricting our attention to some particular semiring types.[4] It is also possible to specialize the composition when the regular model is acyclic (and then finite, as is the case of word graphs). In this case, a topological order can be used to compute the result in a dynamic programming or variable elimination way.

Section 5.6 reviews some common parsing/decoding approaches and formalisms. For the sake of completeness, some well known parsing and decodign algorithms are briefly reviewed as well. A final section summarizes some ideas and draws some conclusions.

5.2 RECOGNITION, PARSING, DECODING

Recognition, parsing and decoding are related and overlapped concepts. Although they could be better defined after the formalisms described in following sections, it is preferable to sketch some general ideas now. Roughly speaking, their main difference is that recognition only checks if a given input belongs to a language or set of accepted inputs, whereas parsing retrieves the set of structures compatible with that input.

Decoding is sometimes related to the "noisy channel" metaphor. In all cases, there is a description of a set of structured data composed by items (sequences, trees, pairs of related sequences, etc.) and the process consists in determining some information from the subset compatible with the observed input, as depicted in Figure 55. For instance, parsing is commonly understood as the problem of retrieving the set of parse structures (parse trees) which are compatible with the observed data. Depending on what is determined, we can even talk of translation.

4 A simple example for the impatient: many computations can be avoided, in a sum of products using the boolean semiring, once a true value is obtained in a partial sum. In the same way, it suffices to find a correct derivation to determine that a sentence can be generated by a grammar.

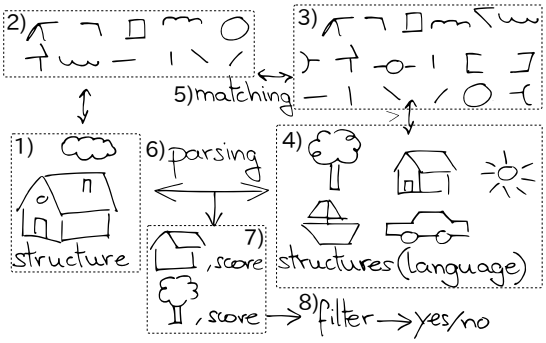


Figure 55: Some ideas related to recognition and parsing: what is observed (1) is structured in such a way that it is composed of a more reduced repertoire of substructures (2 and 3) which can be combined following some constraints (a sequence is a good example). On the other side, we need a model which can be thought as a (possibly infinite) set of structures (4) which is known as a language (described, for instance, by means of a grammar). The parsing process (6) is based on the resemblance of the structure constituents (5) and retrieves the structures of the language which has some resemblance with the observed input together with additional information (7). These structures may include information about how they can be built (e.g. parse trees). The matching process can be partial, which can be used to deal with translation. Different types of (more or less detailed) information can be obtained afterwards (8). Other routes exists and, for instance, recognition does not necessarily require step (7) to attain (8).

There is a clear relationship between parsing and decoding. Some parsing algorithms are just recognizers providing also a set of items which can be used to construct the parse trees. In an abuse of notation, they are also called parsing algorithms.

But not all recognizers can be converted into parsers. Even if they could, some algorithms require to modify the grammars in such a way that the parse trees of the original one cannot be obtained from the transformed version. In this way, even if they can can be used to determine the membership problem, the corresponding parse trees are no longer useful.

Relating decoding, some decoding methods are based on graphical models or on weighted finite state models which, in turn, are commonly associated to parsing. Traditionally, parsing algorithms are more focused on context free models whereas decoding is related to regular languages. Nevertheless, the idea of extending decoding tools to context free models appears very naturally and consequently has been proposed many times, and from long time ago, in the literature.[5] Nowadays, when statistical machine translation coupled with speech recognition is a mainstream topic, it is not unusual to design decoders using formalisms which take reordering and alignments into account.

The rest of this chapter will make use of semirings to model parsing as the composition of weighted transducers. This only covers a particular case of the general idea illustrated in Figure 55, since structures other than sequences are not considered. An example of such an structure not taken into account in our setting is the case of two dimensional structures commonly found in the recognition of printed/handwritten mathematical expressions.

5 And, unfortunately, some ideas appear periodically as novelties.

5.3 WEIGHTED LANGUAGES AND SEMIRINGS

The concepts of recognition, parsing and decoding can cover concepts more general than sequences. The recognition of mathematical formulae, for instance, can be described by means of 2D grammars, to put an example. Our work is limited to sequences. In this setting, the use of formal languages with union and concatenation operations has been proved useful for things such as determining acceptance, obtain parse trees, n-best interpretations, etc. These processes are described in a common way using an algebraic approach. This is one of the main features of what is known as *semiring parsing* [Goodman 1999]. Nevertheless, some *caution* is required when using this term since it is not only limited to the use of semirings to describe the result of a parsing process but is a "pack" which also implies using deductive systems to describe the process itself. It is obvious that semirings can be used for parsing purposes without necessarily resorting to deductive systems.

The free monoid and formal languages

A monoid is an algebraic structure $\langle \mathbb{A}, \otimes, 1 \rangle$ on the set $\mathbb{A}$ with a single associative binary operation $\otimes$ and an identity element 1. A monoid is commutative if $\otimes$ is commutative.

The free monoid on an alphabet Σ is usually denoted by $\Sigma^\star$ and is defined over the set of finite strings composed by elements from Σ using the string concatenation operation. The identity is the empty string usually denoted by ϵ. Formal languages over an alphabet Σ are usually described as subsets of the free monoid $\Sigma^\star$.

Semirings

A semiring[6] is an algebraic data structure similar to a ring without the minus operation. It is a tuple $\langle \mathbb{K}, \oplus, \otimes, 0, 1 \rangle$ such that:

- $\langle \mathbb{K}, \otimes, 1 \rangle$ is a monoid;
- $\langle \mathbb{K}, \oplus, 0 \rangle$ is a commutative monoid;
- $\otimes$ distributes over $\oplus$, i.e. $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ and $c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$ $\forall a, b, c \in \mathbb{K}$;
- 0 is an annihilator for $\otimes$, i.e. $0 \times a = a \times 0 = 0$, $\forall a \in \mathbb{K}$.

A semiring is *idempotent* if $a \oplus a = a$ $\forall a \in \mathbb{K}$. A semiring is *ordered* if there is a partial ordering $\leqslant$ such that $\oplus$ is monotonic. When $\leqslant$ defines a total ordering, the semiring is called *totally-ordered*. $\mathbb{K}$ is *monotonic* when $a \leqslant b \Rightarrow (a \otimes c \leqslant b \otimes c) \wedge (c \otimes a \leqslant c \otimes b)$ $\forall a, b, c \in \mathbb{K}$, whereas it is *superior* if $a \leqslant a \otimes b, a \leqslant b \otimes a$ $\forall a, b \in \mathbb{K}$.

Some semirings are extended to directly describe the unary closure operator $a^\star$. In *complete*[7] semirings, this operator can be defined as $a^\star = \sum_{i \leqslant 0} a^i = 1 + a + a^2 + \cdots$

6 This section is just an sketch. For a good introduction to semirings see [Kuich 1997].

7 Complete semirings are those where infinite sums make sense and are: (1) an extension of finite ones, (2) associative, (3) commutative, and (4) satisfying the distribution laws.

Table 3: Some common semirings. Table inspired by [Mohri *et al.* 2008; Table 1] and by [Goodman 1998; Section 2.2.1]. The *artic* semiring is defined in [Goodman 1998; Section 2.2.1] as the opposite of *tropical* semiring.

Semiring	Set	$\oplus$	$\otimes$	0	1
boolean	$\mathbb{B} = \{\text{true}, \text{false}\}$	$\vee$	$\wedge$	false	true
probability	$\mathbb{R}_+ \cup \{\infty\}$	$+$	$\times$	0	1
tropical	$\mathbb{R}_+ \cup \{-\infty, +\infty\}$	min	$\times$	$-\infty$	0
artic	$\mathbb{R}_+ \cup \{-\infty, +\infty\}$	max	$\times$	$-\infty$	0
log	$\mathbb{R}_+ \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$\times$	$-\infty$	0
real	$\mathbb{R}_+$	$+$	$\times$	0	1
Viterbi	$[0, 1]$	max	$\times$	0	1
count	$\mathbb{N}$	$+$	$\times$	0	1
Viterbi-derivation	$[0, 1] \times 2^{\mathbb{E}}$	$\max_{\text{Vit}}$	$\times_{\text{Vit}}$	$\langle 0, \emptyset \rangle$	$\langle 1, \{\langle\rangle\} \rangle$
Viterbi n-best	$\{\text{topn}(X) \mid X \in 2^{[0,1] \times \mathbb{E}}\}$	$\max_{\text{Vit-n}}$	$\times_{\text{Vit-n}}$	$\langle 0, \emptyset \rangle$	$\langle 1, \{\langle\rangle\} \rangle$

Some common semirings used as weights in the formalisms described below are illustrated in Table 3. Most of them are commutative. The case of non-commutative requires special attention and, indeed, some definitions such as the outer probabilities associated to the inside-outside algorithm of context free languages assume commutative semirings. Some notable non-commutative semirings used in this work are: the semiring of weighed languages and the semiring of weighed transducers.[8]

In this work, semirings are used in a general way before making use of special properties of particular cases to obtain more efficient algorithms: A first general part is mainly devoted to explain what we are trying to obtain. The specialization part will make it clear why some problems become essentially equivalent to shortest paths or to search problems. Hopefully, it will also become clear why it is important to obtain good solutions as fast as possible to apply pruning techniques. The separation of the description into two levels (general and specialization) probably provides a better understanding of the whole picture.

Finally, let us observe that some semirings are defined to include certain features. For instance, the lexicographic semiring [Roark *et al.* 2011] (see Section 9.2) allows an elegant treatment of backing off (see Chapter 6 and Section 9.2) without an ad hoc specialization of the general decoding algorithms. Other semirings, like the derivation forest semiring, make unnecessary to distinguish between recognition and parsing since the computed values already contains information about the derivation trees associated to the derivation of the input value.

An interesting issue we have not found elsewhere is how to reconcile the fact that optimal decoding is a NP-complete problem [Casacuberta and de la Higuera 1999] while it has a polynomial cost in terms of semiring operations. The reason of this gap must lie in the cost of actually implementing these operations for certain semiring types.

8 Indeed, two different descriptions of semiring associated to weighed transducers, which differ in how $\otimes$ is defined, are described in this work.

Formal power series, weighted languages and transducers

Formal languages over an alphabet Σ can be alternatively specified by providing the indicator function from $\Sigma^\star$ to the boolean semiring:

$$1_L(s) = \begin{cases} \text{true}, & \text{if } s \in L \\ \text{false}, & \text{otherwise.} \end{cases}$$

This allows for a direct generalization to the case of weighted languages by replacing the boolean with other semirings. Indeed, it turns out that functions $f : \Sigma^\star \to \mathbb{K}$ are known as *formal power series* and formal languages are a particular case when $\mathbb{K}$ is the boolean semiring.

When the domain of the weighted function is not the free monoid we can more generally talk about just weighted sets. When this set is a Cartesian product[9] $\mathcal{X} \times \mathcal{Y}$, it can be considered as a weighted transducer associating values to pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ which, in turn, can be further generalized to a weighted n-ary relation by means of a weighted set over $A_1 \times \cdots \times A_n$. In our context, only languages and n-ary relations of languages are required. Generator and recognizer devices acting on n-ary relations of languages are also known as multitape devices. Traditional descriptions distinguish an especial input tape. This is specially relevant for the case of binary relations where transducer composition, defined below, require a matching between the output of a transducer and the input of the other. Some weighted transducers are partial functions meaning that, for each value x, there is at most one value y such that (x, y) is assigned a non-zero value.

common trick

A trick which is useful in some cases consists in considering languages as identity transducers. In this way, a weighted language L defined over $\Sigma^\star$ can be considered equivalent to a weighted transducer L' defined over $\Sigma^\star \times \Sigma^\star$ as follows:

$$L'(s, s') = \begin{cases} L(s), & \text{if } s = s' \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

marginalization

Conversely, it is possible to obtain what are called the *domain* and the *range* weighted languages of a weighted transducer by marginalization.

Stochastic languages are another case of weighted languages. They describe probability distributions over the elements of $\Sigma^\star$. Note that it does not suffice the use of a certain semiring (e.g. the probability semiring) to obtain a stochastic language. Certain weighted language formalisms admit sufficient conditions to assure that the corresponding weighted language is a proper stochastic model.

Transducer composition

The concept of transducer composition, usually defined for weighted finite state transducers (WFST) [Mohri *et al.* 2002], can be defined in general terms over weighted relations [Dyer 2010]. Given two weighted

9 The term "binary relation" is also used in some texts, although others assume that this implies that both sets are the same, which is not necessarily the case.

functions $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{K}$ and $g : \mathcal{Y} \times \mathcal{Z} \to \mathbb{K}$, the composition $f \circ g$ is defined as:

$$(f \circ g)(x, z) = \bigoplus_{y \in \mathcal{Y}} f(x, y) \otimes g(y, z) \tag{5.2}$$

This operation, essentially similar to matrix multiplication,[10] is associative. The idea is that the "common tape" (the codomain of f and the domain of g, denoted by $\mathcal{Y}$ in the example) of two transducers are matched and then combined (with $\otimes$). The final result is marginalized (with $\oplus$). This has some reminiscences with the scheme of Figure 55.

Observe that the particular case of the boolean semiring, with the use of identity transducers, leads to the well known approach of *parsing as intersection* which can be traced back at least to [Bar-Hillel *et al.* 1961; Lang 1988; 1994]. In this paradigm, the process of recognizing an input sequence (to determine if this sequence belongs to a language) is converted into the computation of the intersection of the language composed just by this sequence and the language generated by the grammar. This can be generalized to the weighted case by using other types of semirings and other type of transducers. This approach is very general since it not only provides a direct generalization from the case of sentences to the case of word graphs, as is our case, but also allows the computation of prefix and infix probabilities by replacing the input string with the languages of prefixes, infixes, suffixes or another concept expressed by means of languages.

parsing as composition

Weighted languages and transducers are values

There exist many different formalisms to represent (possibly infinite) weighted languages and transducers in a finite way. Some of them, such as formal grammars, automata and finite state transducers are described in Section 5.4. They can be considered, in general, as generative or as recognition devices acting over individual strings (or pairs of strings, in the case of transducers). But this is not the only point of view since we can also consider weighted languages and weighted relations themselves as *values* which can be properly manipulated or combined with union and concatenation operations. In this way, some of these formalisms can be interpreted as systems of equations over the set of weighted languages. This perspective is not at all new since their roots date at least from [Ginsburg and Rice 1962]. For instance, context free languages (described below) are the least solutions of a particular type of equations based on two different types of operation: union and concatenation of languages.

languages and transducers can be values

In formal language theory, the union of two languages L_1 and L_2, over the same free monoid $\Sigma^\star$, is defined as the set theoretic union:

$$L_1 \cup L_2 = \{s \mid s \in L_1 \vee s \in L_2\} \tag{5.3}$$

which is generalized to the weighted case as the weighted union:

$$(L_1 \oplus L_2)(s) = L_1(s) \oplus L_2(s) \tag{5.4}$$

10 Indeed, matrices can be seen as weighted relations.

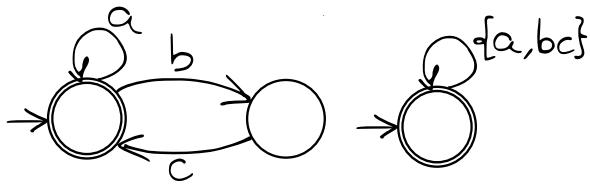


Figure 56: Automata using individual labels (left) and the equivalent automaton using language values (right).

Another interesting operation is the concatenation of languages:

$$L_1 \cdot L_2 = \{xy \,|\, x \in L_1, y \in L_2\} \tag{5.5}$$

whose extension to the case of semirings can be defined as follows:

$$(L_1 \cdot L_2)(s) = \bigoplus_{x,y \in \Sigma^\star, s = x \oplus y} L_1(x) \otimes L_2(y) \tag{5.6}$$

Some algebraic structures can be derived from others and semirings are not an exception. An example is the semiring defined over matrices. In a similar way, we can found the semiring of formal languages [Ésik and Kuich 2007] which can be extended to construct a semiring over the set of weighted languages where:

- the zero value is the empty set (the weighted language assigning zero to all possible strings);
- the identity is the unit language $\{\epsilon\}$ whose only non-zero value is the empty string which has value 1;
- the $\oplus$ operator is the weighted union; and finally
- the $\otimes$ operator is the language concatenation.

Figure 56 describes an automata using individual labels and the equivalent representation using language values.

In a similar way, the set of weighted transducers can be equipped with $\oplus$ and $\otimes$ operations to constitute a semiring. Given two transducers $f, g : \Sigma^\star \times \Omega^\star \to \mathbb{K}$, these operations are defined as expected:

$$(f \oplus g)(x, y) = f(x, y) \oplus g(x, y) \tag{5.7}$$

and

$$(f \otimes g)(s, t) = \bigoplus_{\substack{x,y \in \Sigma^\star, s = x \oplus y \\ r,k \in \Omega^\star, t = r \oplus k}} f(x, r) \otimes g(y, k) \tag{5.8}$$

The identity element is the transducer which assigns the identity of $\mathbb{K}$ to the pair (ϵ, ϵ), usually denoted $\epsilon : \epsilon$, and zero to the rest. These operations can be extended to n-ary relations.

alternative semiring definition

An alternative way to define a semiring over the set of weighted transducers, when they are defined over the same alphabet (i.e. over $\mathbb{K}^{\Sigma^\star \times \Sigma^\star}$) uses the transducer composition as the product operator $\otimes$.

The identity element would be the identity transducer which assigns value 1 to pairs $(s, s), s \in \Sigma^\star$ and 0 to the rest.

Some additional operations defined over languages are the left- and right-quotients. Given two languages L_1 and L_2 defined over the same free monoid $\Sigma^\star$, the left-quotient is defined as:

$$L_1 \backslash L_2 = \{x \mid \exists y \in L_1, yx \in L_2\} \tag{5.9}$$

similarly, the right-quotient is defined as:

$$L_1 / L_2 = \{x \mid \exists y \in L_1, xy \in L_2\} \tag{5.10}$$

Note that these quotients "lose information" in the sense that they cannot be reversible in general, as shown in the following example $(\{a, b\} \cdot \{c, \epsilon\})/\{c, \epsilon\} = \{ac, bc, a, b\}/\{c, \epsilon\} = \{a, b, ac, bc\}$, meaning that $(YX)/X \neq Y$ in general. Similarly, $X\backslash(XY) \neq Y$ in general. But note that this may be valid in some cases: $\{w\}\backslash(\{w\}Y) = Y, \ \forall w \in \Sigma^\star$.

Unfortunately, the manipulation of weighted languages and transducers as algebraic structures do not allow an *easy* definition of the transducer composition in terms of more elementary parts. In particular, transducer composition does not distribute over concatenation in general. The language concatenation operation is not compatible with the left- and right-quotients either, as pointed out before. That is probably why many works defining the transducer composition operation propose algorithms which *are not derived from the definitions*. They are usually defined for the particular case of finite state transducers and given as is. A contribution of this work is the derivation of a transducer composition algorithm to combine a regular with a context free model from the previous definitions. We believe that this is not only of anecdotal interest: it will be shown that transducer composition of finite state models is a particular case of the composition of regular and context free models, which is the most general case of composition considered in this work.[11] In order to better explain the derivation of this algorithm, we need first to present some formalisms.

5.4 SOME FORMALISMS

This section briefly reviews some formalisms. Since our work is restricted to (weighted) regular and context free languages, many approaches such as tree adjoining grammars, link grammars, synchronous CFGs, etc. have been left aside on purpose.

5.4.1 Formal/generative grammars

Typical introductions to parsing technologies start by describing formal models associated to the well known Chomsky–Schützenberger hierarchy: families of subsets of the free monoid over a finite alphabet characterized by the machinery which can recognize or generate strings belonging to these languages. Although there are many types

11 We have consciously left aside other formalisms such as synchronous CFGs which may be of interest for other more complex tasks such as SMT.

of generator or recognition devices, it is usual to start by describing formal grammars. A formal grammar $G = (N, \Sigma, P, S)$ is described by means of:

- a finite alphabet Σ, the language described by the formal grammar is made of these symbols. These symbols are known as "terminal" because they constitute the symbols of the produced language;
- an additional finite alphabet of "intermediate", "auxiliary" or, more commonly, "non-terminal" symbols N, which is disjoint from Σ; in some cases Z is used to denote $\Sigma \cup N$;
- a finite set P of rewrite or production rules of the form $u \to v$ where both u and v are strings of $(N \cup \Sigma)^\star$. Such a rule can be used in a rewriting step which modifies a sequence from $(N \cup \Sigma)^\star$. This sequence is known as "sentential form" and the rewriting step (denoted by $\longrightarrow$) using the rule $u \to v$ replaces an occurrence of substring u in the sentential form by the string v. Grammars can be classified into families depending on the restrictions on the type of rules;
- a special or distinguished non-terminal symbol $S \in N$ known as the start symbol or the axiom.

Formal grammars are based on the concept of rewriting strings, as explained before. A derivation, represented by $\xrightarrow{\star}$, is a sequence of zero or more rewriting steps. Similarly, $\xrightarrow{+}$ denotes a sequence of one or more rewriting steps.

The language produced by the grammar is defined as the set of strings composed by terminal symbols which can be obtained, by derivation, from the start symbol S:

$$L(G) = \{w \in \Sigma^\star \mid S \xrightarrow{\star} w\} \tag{5.11}$$

Chomsky–Schützenberger hierarchy

The Chomsky–Schützenberger hierarchy[12] defines classes of formal grammars organized in a set of strictly nested levels (as depicted in Figure 57) depending on the restrictions applied to the rules. Note that this hierarchy is defined here for the un-weighted case:

UNRESTRICTED grammars have no restrictions on the set of production rules. They generate the set of recursively enumerable languages;

CONTEXT-SENSITIVE grammars rules restricted to the form $\alpha A \beta \to \alpha \gamma \beta$ where $A \in N, \alpha, \beta, \gamma \in (N \cup \Sigma)^\star$ and $\gamma \neq \epsilon$ (an exception is possible if ϵ belongs to the language);[13]

12 See also, for instance, `http://en.wikipedia.org/wiki/Chomsky_hierarchy`.

13 The rule $S \to \epsilon$ is allowed if ϵ belongs to the language, S is the axiom (the only initial non-terminal symbol) and S does not appear in the right side of any rule.

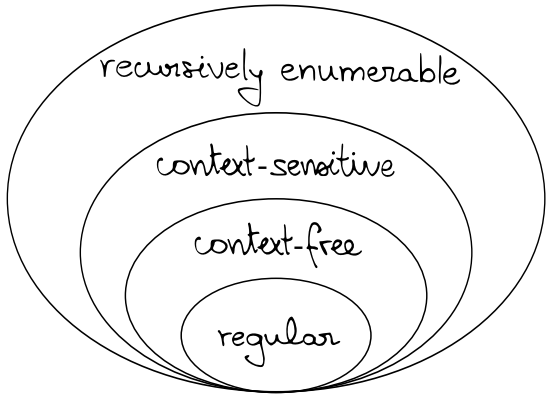


Figure 57: Chomsky–Schützenberger hierarchy.

CONTEXT-FREE grammars correspond to the rules $A \to \gamma$, where $A \in N$ and $\gamma \in (N \cup \Sigma)^\star$. The languages generated by these grammars are known as context-free languages and correspond also with the languages generated by non-deterministic push-down automata, by context-free expressions [Winter *et al.* 2011; Section 5], by extended CFGs [Madsen and Kristensen 1975] or even by recurrent-transition networks (RTNs) [Woods 1970];

REGULAR grammars are those that describe regular languages, which are those languages which can be expressed by means of regular expressions[14] or, alternatively, by finite state automata (described below). They constitute a proper subset of context free grammars. Some CFG types guarantee that the resulting language is regular, as is the case of left-linear and right-linear grammars (see below), but this condition is not necessary and the general problem of determining whether a general GFG is regular or not is undecidable [Hopcroft and Ullman 1979].

Let us remark that there are many other language families, and some of them do not fit in this scheme.

From now on, we will only work with (weighted) context-free and with regular languages and models.

(Weighted) Context-free grammars

CFGs have many peculiarities worth studying. This work does not pretend to be a review, but just to introduce the required information to describe some ideas related to parsing algorithms.

tree traversal

Derivations associated to context free grammars replace, in each step, a non-terminal symbol by the right-hand side of a rule associated to it. The sequence of derivations leading to a sequence of terminal symbols can be represented by means of a derivation tree where internal nodes are non-terminal symbols and the rule applied on this

14 Regular expressions are expressions made of constants (to describe the empty set, the empty string and literal symbols) and operators (to denote concatenation, alternation and the Kleene star). An extension called context free expression replace the Kleen star by a general fixed-point operator μ and allows the definition of CFGs.

symbol determines its children. A tree traversal with a proper ordering of leaves retrieves the final string by discarding internal nodes.

A grammar which produces at least one string with more than one parse tree is called ambiguous. Note that some languages admit both ambiguous and unambiguous grammars but others, known as inherently ambiguous, can only be modeled by means of ambiguous grammars.

Several derivation sequences can lead to the same derivation tree depending on the order rules are applied. Indeed, two particular derivation sequences are usually highlighted: *leftmost* and *rightmost* derivations. In a leftmost (respectively, rightmost) derivation, the variable replaced in each step is the leftmost (respectively, rightmost) variable of the sentential form (that is, this variable has no other variable to its left (respectively, to its right)).

leftmost derivation

It is possible to extend the definition of CFGs to the weighted case, leading to WCFGs. To this end, each production rule is associated a weight $k \in \mathbb{K}$. In some cases, the use of the start symbol S is replaced by an initial weight function from the set of non-terminal symbols N to $\mathbb{K}$. The weight of each string is defined as a sum of products of weights associated to different leftmost derivations leading to the string. Note that the requirement of using leftmost derivations is not mandatory, it is just a way to ensure that the different ways to derive a sequence ignore the order the rules are applied. This concept could have been defined in terms of parse trees. Note also that the order these weights are multiplied is important in the case of non-commutative semirings, which has some consequences in some definitions as is the case of the "outside" value described below.[15]

sum of products

Let us remark that the value associated to each sentence is a (large) sum of products and that this notion appears in many other problems as is the case of graphical models (factor graphs, etc. see Section 4.1.1). The distributive properties of semirings allow clever ways to reduce the computational cost w.r.t. naive implementations. This reduction is general for any semiring type, but note that further reductions can be achieved when taking particularities of certain semirings into account. We will focus on general reduction techniques for the moment and delay semiring specific optimizations to Section **??**.

Probabilistic CFGs (PCFGs) constitute a relevant particular case of those WCFGs whose values are positive values and can, hence, be interpreted as probabilities. A WCFG is proper if each production is associated a positive real-valued weight which can be interpreted as the conditional probability of choosing this rule when expanding the non-terminal.[16] A WCFG is convergent when the sum of those values for all finite strings converge to a finite number. A PCFG is a convergent WCFG which is also consistent (converges to a probability mass of one). Note that some proper WCFG may not be consistent since they assign some non-null probability mass to infinite derivation sequences. Conversely, some consistent grammars are not necessarily proper [Nederhof and Satta 2008]. It turns out that most PCFG estimation techniques produce consistent grammars.

15 To our knowledge, this fact is not clearly specified in the literature, probably because nobody is interested in non-commutative semirings?

16 The weights of productions associated to each nonterminal are constrained to sum 1.

Some sub-classes

Let us describe some sub-classes of CFGs. There are many sub-classes of CFGs which are not reviewed here: for instance, LR(k) grammars associated to deterministic pushdown automata or LL(k), the class of grammars parsable by top-down recursive descent parsers. They are not reviewed here because we are mainly interested in some particular aspects related to some general parsing algorithms which can be adapted to weighted CFGs.

Two sub-classes which are mentioned or used below are:

BRACKETED or Dyck languages are those formed by well-balanced parenthesis. Interestingly, any context free language can be expressed as the intersection of a regular and a Dyck language (Chomsky-Schützenberger theorem[17] [Chomsky and Schützenberger 1963]). This idea is exploited for parsing purposes in [Hulden 2011];

LINEAR grammars are those who have at most one non-terminal in the right hand side of productions. There are two particular types of linear grammars:

EXTENDED RIGHT LINEAR if all production rules are right-linear. A production rule is right-linear when the non-terminal of the right-hand side (in case there is one) is at the right. Rules are of the form $A \to \alpha B$ or $A \to \alpha$, where $\alpha \in \Sigma^\star$, $A, B \in N$. A grammar is right linear (without *extended*) when rules are of the form: $A \to aB$, $A \to a$ or $(A \to \epsilon)$, where $a \in \Sigma$ and $A, B \in N$. The last rules (without B) are called terminating;

EXTENDED LEFT LINEAR and left-linear are similar to the right-case by replacing the "at the right" with "at the left", meaning that the non-terminal of the right-hand side has no other symbol on its left.

As explained before, right- and left-linear grammars lead always to regular grammars. The condition is sufficient but not necessary[18] and, indeed, a grammar mixing left and right linear rules do not necessarily lead to a regular language.[19]

Normal forms

There are several normal forms associated to context-free grammars. A normal form tries to restrict the variability of the description of the grammar while preserving the generated language (but note that the structure of parse trees may change). A normal form has to satisfy two properties: 1) the languages generated by the set of grammars in a given normal form has to be as powerful as the unrestricted system and 2) in order to be useful, a general construction algorithm to convert a grammar to the equivalent in the normal form has to be available.

parse structure not preserved

17 http://en.wikipedia.org/wiki/Chomsky%E2%80%93Sch%C3%BCtzenberger_theorem.

18 As it has been pointed out before, determining whether a grammar leads to a regular language or not is an undecidable problem.

19 For instance, the language $\{a^n b^n \mid n > 0\}$, which is not regular, can be specified with a linear grammar mixing left- and right-linear production rules.

Some popular normal forms are:

CHOMSKY normal form (abbreviated CNF): is a particular type of bilinear grammar. Bilinear grammars allow up to two non-terminals in the right-hand sides of production rules. As we will see, the cost of some parsing algorithms depends on the rank of the grammar (the maximum length of the right-hand-side of rules, this is specially clear with Unger's algorithm described in Section 5.6) and the lower the rank the better. This explains the interest in the binarization of grammar rules. The CNF is perhaps the most prominent way to achieve this binary-rule property.

The most usual and strict version of CNF only allows rules of the form $A \to a$ and $A \to BC$, where $A, B, C \in N$ and $a \in \Sigma$, with the possible exception of the rule $S \to \epsilon$ where S is the initial symbol and, in this case, S cannot be used recursively.

The practical importance of CNF comes from the popularity of CYK parsing algorithm (also described in Section 5.6) which uses grammars in this normal form, although some less restrictive variants can be found. The reader is referred to [Lange and Leiß 2009] for a good review on the different variations of this form, the associated versions of CYK parsing and the different possibilities and issues relating the transformation of CFGs into CNFs;

CANONICAL TWO FORM or just 2NF is another type of bilinear grammar closely related to CNF. Rules of a grammar in 2NF are of the form $A \to \alpha, \alpha \in (N \cup \Sigma)^\star, |\alpha| \leqslant 2$. This normal form is usually employed as an intermediate step in the process of conversion into CNF [Lange and Leiß 2009]. Section 5.4.3 presents a normal form for RTNs closely related to (a subset of) 2NF;

GREIBACH normal form (abbreviated GNF): only allows productions of the form $A \to a\alpha$, where $a \in \Sigma$ and $\alpha \in (N - \{S\})^\star$, with the possible exception of the rule $S \to \epsilon$. It can be proved that any CFG can be converted into GNF. One advantage o this normal form is that it is free from left-recursions and that each rule (excepting $S \to \epsilon$) consumes a terminal symbol. A grammar is left-recursive if there is a non-terminal A such that $A \xrightarrow{+} A\alpha$, where $\alpha \in (N \cup \Sigma)^\star$. There are other left-recursion removal procedures less restrictive than a GNF transformation [Moore 2000].

Matrix form and systems of equations

CFGs can be represented as a system of equations [Ginsburg and Rice 1962; Chomsky and Schützenberger 1963; Rosenkrantz 1967] with one equation per non-terminal defined as the sum of all right hand sides of its production rules. This can be viewed as if:

- each non-terminal represents or defines a language;
- the set of rules associated to the same non-terminal (in the left-hand side) defines an equation using language union and language concatenation;
- the language defined by the grammar corresponds to the language associated to the axiom S.

For instance, the following grammar $G = (\Sigma, N, P, S)$:

$$S \to aA$$
$$A \to B|SB$$
$$B \to b$$

would correspond, with an abuse of notation, to the following system of equations:

$$S = aA$$
$$A = B + SB$$
$$B = b$$

although it is more correct to consider a system of equations defined over languages in the formal power series (or weighted language) semiring:

$$S = \{a\} \otimes A$$
$$A = B \oplus (S \otimes B)$$
$$B = \{b\}$$

If we consider that the set of variables or non-terminals is a vector $[S, A, B] \approx (\mathbb{K}^{\Sigma^\star})^{|N|}$ or a mapping $N \to \mathbb{K}^{\Sigma^\star}$ from the set of non-terminals N to the space of weighted languages (i.e. to formal power series), the system of equations can be described in the form $x = f(x)$. It is clear that a solution is a fixed point of f. A solution is a *minimal* fixed point if $f(A) \subseteq g(A)$ for every $A \in N$ and for every solution g.

minimal fixed point

It can be proven that the minimal fixed point solution is unique and coincides with the previous definition given in terms of sums of derivations [Aho and Ullman 1972; Ésik and Kuich 2007].

The systems of equations describing a grammar can also be represented in matrix form [Rosenkrantz 1967] in at least two different ways:

$$\begin{bmatrix} S & A & B \end{bmatrix} = \begin{bmatrix} S & A & B \end{bmatrix} \otimes \begin{bmatrix} \emptyset & B & \emptyset \\ \emptyset & \emptyset & \emptyset \\ \emptyset & \{\epsilon\} & \emptyset \end{bmatrix} \oplus \begin{bmatrix} \{a\} \otimes A & \emptyset & \{b\} \end{bmatrix}$$

and

$$\begin{bmatrix} S \\ A \\ B \end{bmatrix} = \begin{bmatrix} \emptyset & \{a\} & \emptyset \\ \emptyset & \emptyset & \{\epsilon\} \oplus S \\ \emptyset & \emptyset & \emptyset \end{bmatrix} \otimes \begin{bmatrix} S \\ A \\ B \end{bmatrix} \oplus \begin{bmatrix} \emptyset \\ \emptyset \\ \{b\} \end{bmatrix}$$

depending on our desire to remark rules which do not start by a non-terminal, in the first case, or to group rules which end by the same non-terminal, on the other case. In both cases, the elements of the matrix and the last (row or column) vector are expressions involving products and summations of variables and semiring values. The first approach has some interest in some procedures for obtaining the GNF of a CFG. The second one is more relevant for the case of right-linear grammars.

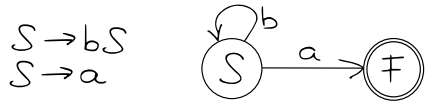


Figure 58: Example of the conversion of a right linear grammar into the corresponding automaton.

5.4.2 Finite state automata and transducers

The last matrix form described before can be written as $x = Mx + b$. This has some interesting particularities when applied to right-linear grammars since both the matrix M and the column vector b only contain semiring values. The solution of this linear equation is of the form $x = M^\star b$ where $M^\star$ represents the closure of the matrix, which can be defined[20] as $M^\star = I + M + M^2 + \cdots = \sum_{i>=0} M^i$, being "I" the identity matrix. Matrix M can be represented as a graph and, by the inclusion of an additional vertex to include the information of b, we can obtain the graphical representation of a finite state automaton describing the same language as the original grammar:

- each non-terminal corresponds to an state of the automaton, the initial state is associated to the axiom S, an additional final state $F \notin N$ is also included;
- each production rule of the form $A \to aB$ corresponds to a labeled transition from state A to state B, being a the label of this transition;
- production rules $A \to B$, when they are allowed, correspond to null-transitions;
- production rules of the form $A \to a$ correspond to transitions to the final state F.

An example is shown in Figure 58. The language associated to each non-terminal A corresponds to the sum of all paths from A to F in such a way that:

- labels of the edges of the path are semiring values which are to be multiplied using the semiring operator $\otimes$;
- the value associated to each path is accumulated using the semiring operator $\oplus$.

A particular case is the language associated to the axiom S, which is the language associated to the grammar and, in this case, to the automaton. That is the reason why S is considered the initial state.

[Ésik and Kuich 2007] also defines automata in a very generic way in terms of a transition matrix M (defined as a mapping from $Q \times Q$) and an initial S and a final P state vector (functions from Q to $\mathbb{K}$). The *behavior* of the automata is given in terms of $SM^\star P$ in a way similar to [Rosenkrantz 1967].

20 Some woks propose a direct description of this closure operation in the axiomatic definition of the semiring structure, in terms of some properties it displays. Note also that not all semirings have a well defined $\star$ since this series must converge to a semiring value.

It is possible to remove the need of initial and final state vectors by augmenting the matrix, by including the final state F in M:

$$M' = \left[\begin{array}{c|c} M & b \\ \hline \emptyset \cdots \emptyset & \emptyset \end{array}\right]$$

We can directly obtain the language associated to each non-terminal in the last row of $(M')^\star$ as illustrated in the example of Figure 58 which corresponds to the following right-linear grammar:

$$S \to bS$$
$$S \to a$$

where we can obtain $M' = \begin{bmatrix} \{b\} & \{a\} \\ \emptyset & \emptyset \end{bmatrix}$ so that:

$$M^2 = \begin{bmatrix} \{b\} & \{a\} \\ \emptyset & \emptyset \end{bmatrix} \begin{bmatrix} \{b\} & \{a\} \\ \emptyset & \emptyset \end{bmatrix} = \begin{bmatrix} \{bb\} & \{ba\} \\ \emptyset & \emptyset \end{bmatrix}$$

meaning that S contains b and ba which can be obtained, respectively, following paths of length 1 and 2 in the automaton. The language defined by the automaton would correspond to the value of the first row last column of $(M')^\star$, i.e. the regular expression $b^\star a$.

We saw that *extended* right-linear grammars are a generalization allowing rules of the form $A \to \alpha B$ or $A \to \alpha$, where $A, B \in N$ and $\alpha \in \Sigma^\star$. This can be generalized to the weighted case by assuming $\alpha \in \mathbb{K}$ and using the formal power series semiring (a.k.a. "weighted language semiring"). The set of elements from $\mathbb{K}$ used to define those models should be restricted to those weighted languages whose support is a finite set of strings in order to avoid the trivial definition of any weighted language. This formalism allows an easy generalization from acceptors to transducers since we only need to change the semiring type and label the transitions accordingly. The generalized weighted transducers and acceptors using the convention relating the finite support of underlying weighted languages can be easily converted to the previous more restrictive definition which only allows the use of terminal symbols or ϵ to denote null transitions: it suffices to add intermediate states to split certain transitions labeled with languages composed by several strings or by strings with more than one symbol, as illustrated in Figure 59.

Although the definition of weighted finite state transducers over a semiring $\mathbb{K}$ varies in the literature, all of them have common points such as the use of a finite set of states Q. Most definitions other than those in [Ésik and Kuich 2007] define WFSTs in terms of input and output alphabets. They usually differ in how initial states are defined or where and how outputs are encoded. For instance, [Mohri *et al.* 2002] defines WFSTs as a tuple $T = \langle \Sigma, \Omega, Q, E, i, F, \lambda, \rho \rangle$ where:

- Σ is a finite alphabet called the input alphabet;
- Ω is another finite stated known as the output alphabet;
- Q is a finite set of states;
- E is a finite set of transitions $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times \mathbb{K} \times Q$ which specify how an arc from an origin or source state to a

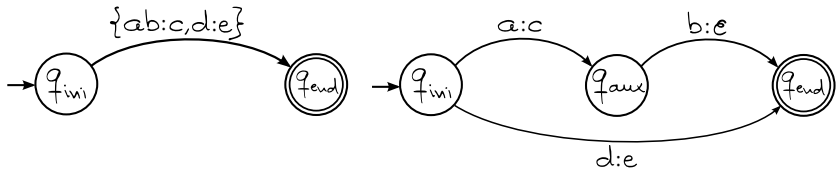


Figure 59: Splitting a transition labeled with a weighted language into transitions using only symbols or ϵ: (on the left) a finite state transducer where transitions are labeled with elements from $\Sigma^\star \times \Omega^\star$ (sets of pairs of input/output strings) and (on the right) an equivalent automaton where transitions can only be labeled with elements from $(\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\})$, an auxiliary state is included to split the string ab.

destination state is labeled with input and output symbols (or the lack of a symbol) as well as a semiring weight;

- $i \in Q$ is an initial state;
- $F \subseteq Q$ is the set of final states;
- λ is the initial weight; and
- ρ is a final weight function.

Other similar definitions (e.g. [Dyer 2010] or also in wikipedia[21]) prefer a set of initial states and may include or avoid initial and final weight functions since they can be replaced by the existence of null transitions. It is even possible to specify just an initial and a final state. Other definitions define transitions in such a way that the output labels of transitions may be strings (elements of $\Omega^\star$). All these formalisms can be converted into the previous version based on a transition matrix explained before.

WFSTs are very important in our area of research due to the fact that most language models used in ASR, HTR and other relevant tasks are usually stochastic regular models (and, more particularly, n-grams), this issue will be discussed into more detail in Chapter 6. Among other reasons for explaining the practical success of these models, we can mention the fact that regular models are closed under operations such as union, intersection, the Kleene closure, etc. Also, there are very efficient algorithms for transducer composition, projection, etc.

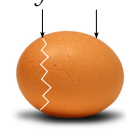
Some operations and properties which are easily determined for regular languages are even undecidable in the case of context free languages. For instance, the intersection of two regular languages is a regular one, but the intersection of two CFGs is undecidable [Hopcroft and Ullman 1979]. Fortunately, the intersection of a regular and a context free model is context free [Bar-Hillel *et al.* 1961]. Moreover, there are constructive methods to compute the resulting model [Bar-Hillel *et al.* 1961]. This algorithm has been extended to the weighted case [Nederhof and Satta 2003; Dyer 2010]. Our approach, described in Section 5.5, has resemblances with this algorithm but is based on RTNs (instead of CFGs). One of the advantages of using RTNs is to make more evident the idea that this can be viewed as an extension of WFST composition. Let us first introduce RTNs into more detail.

21 See `http://en.wikipedia.org/wiki/Finite_state_transducer`.

5.4.3 Recurrent transition networks

Recurrent transition networks [Woods 1970] are formal models that can be represented by means of directed graphs in a very similar way to finite state automata. The basic difference between RTNs and FSA is that RTN edges can also be labeled with model references and not only with terminal symbols (or the aforementioned extension to semiring values). Some works describe these references as states of the same RTN, whereas most other works seem to assume that there is a family of independent finite state models generating a language over $\Sigma \cup N$ each one. In the last case, model references are symbols of N. We would follow the first less conventional approach which, in our humble opinion, leads to a much more elegant formalism.

Mealy vs Moore



We can find in the literature descriptions of RTNs where labels and references are defined in the nodes [Thomae 2006] whereas others prefer to label arcs. This is the Mealy vs Moore discussion we can also found in the finite state model literature.[22]

RTNs are sometimes described as an extension of finite state models where *some recursive mechanism is included*. Traditionally, the behavior of model references resembles a *procedure call* commonly found in imperative computer languages with a stack of returns as found, for instance, in [Han and Choi 1994]. This is not very different from a non-deterministic pushdown automaton. We would like to avoid the procedural definitions usually found in the description of many parsing and decoding algorithms which, in the case of RTNs, naturally leads to the expansion approach commonly found in most RTN implementations. That is why instead of adopting those procedural point of views we rather prefer to see RTNs in the same way as we also see FSA and CFGs: as systems of equations associating weighted languages to a set of variables. The case of graphical representations such as FSAs and RTNs make the relationship between spans and associated semiring values more clear:

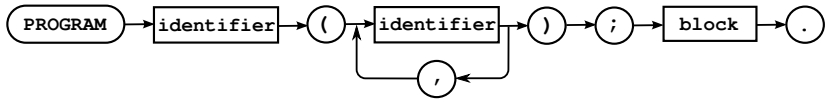
RTNs as systems of equations

- paths are associated to concatenation of edges which are labeled with language values; whereas
- nodes represent points of summation since they can be considered as accumulators for all paths incoming to them.

The case of cyclic structures can be solved by means of the fixed point iteration method, which is also related to the computation of the transitive closure as it was explained when describing right-linear CFGs in the form of a linear system of equations.

Historically, RTNs were probably obtained for pragmatical reasons by merging the right-hand sides of all the rules associated to a given non-terminal of a CFG and by representing this set in the form of an automaton with symbols from $N \cup \Sigma$. Not surprisingly, many parsing algorithms construct some type of finite state device from the grammar or transform it into a pushdown automaton. According to [Woods 1970; Section 6] the idea of converting a CFG into a RTN can be traced back at least to 1963 (referring to [Conway 1963]).

22 Not different from the big endian vs little endian from the Lilliput episode of Gulliver's Travels. We are decidedly in favor of using labels in transitions (Mealy machines).




```
program ::= "program" identifier "(" identifier { "," identifier } ")" ";" block "." .
```

Figure 60: Example of a syntax/railroad diagram (top) and the corresponding definition in EBNF (bottom). This diagram corresponds to a part of the grammar of pascal as found in [Jensen *et al.* 1975].

RTNs are related to *extended context free* grammars (ECFGs) which constitute a representation similar to CFG where right-hand-sides of rules are *regular expressions*.[23] ECFGs are also known as regular right part grammars (RRPGs) in some contexts, but others (e.g. [Hemerik 2009]) define RRPGs as those having automata in the right-hand side of productions, which is the usual definition of RTNs. In any case, ECFGs are possibly known mostly due to the existence of the Extended Backus Naur Form (EBNF) which is a notation to describe CFGs. ECFGs allow the use of some operators on the right-hand side of rules to express repetitions, optionality, exceptions, etc. and are very popular to describe the grammars of some programming languages and protocols,[24] or the formal grammar of XML. The relationship between ECFGs and RTNs is best illustrated by the use of syntax/railroad diagrams to graphically represent EBNF grammars, probably first used by [Jensen *et al.* 1975], as shown in Figure 60.

same expressive power

It is obvious that all three formalisms have the same expressive power since they can be converted from one another (there are several ways to convert a regular expression into automaton, an automaton can be converted into a right-linear CFG and a similar procedure can be applied to RTNs). We will provide a way of describing a normal form of CFGs which plays the same role for RTNs that right-linear languages for finite state automata.

RTNs have been used and extended in many ways and with different purposes in the literature. They are described in [Woods 1970] to define augmented transition networks (ATNs) which are essentially RTNs annotated with conditions on the arcs and supplied with registers or slots to construct structures as the parse process proceeds.[25]

RTNs proposed in [Blanc *et al.* 2005] are "decorated" with functional equations in order formalize linguistic phenomena such as agreement, leading to a formalism similar to Lexical Functional Grammar models [Kaplan and Bresnan 1982].

Relating the use of RTNs in this work, we do not advocate for completely avoiding other representations such as CFGs, but rather to show that the RTNs described here constitute an alternative representation of CFGs (of a specific normal form described below) and to highlight the interest in RTNs in many cases where their potential use seems completely ignored. There are several motivations for using RTNs as a formalism to represent context free languages. On the one side, RTNs seem easier to understand, as can be observed by the use of

23 Briefly defined before when introducing regular models in the Chomsky-Schützenberger hierarchy.

24 Protocols are often described using Augmented BNF (a variation of EBNF).

25 Indeed, ATNs are Turing complete and will not be considered in this work.

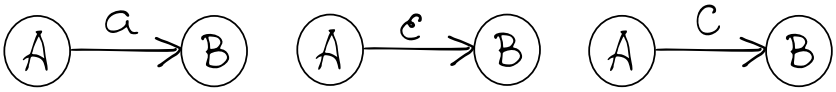


Figure 61: RTN transitions as CFG rules: (left) transition labeled with a terminal symbol corresponds to rules $A \rightarrow aB$, (middle) transition labeled with the null-string ϵ correspond to rules $A \rightarrow B$ and (right) transition labeled with a model reference correspond to rules of the form $A \rightarrow CB$.

syntax/railroad diagrams described before (see Figure 60) or the use of visual editors allowing the graphical description of RTNs in linguistic platforms based on finite state automata and RTNs (as Unitex or Outilex [Blanc and Constant 2006]). On the other side, this representation makes the relationship between many context free and regular algorithms more evident. Indeed, RTNs allow us to avoid many auxiliary definitions and artifacts such as, for instance, dotted rules.[26] It is even possible to directly apply some regular automaton procedures to RTNs. For instance, some RTNs are compacted by minimizing the language represented by each finite state model interpreting the model references as ordinary symbols.[27] The conversion of grammars in EBNF notation into RTNs may benefit from algorithms to convert regular expressions into automata where it is possible to obtain loops which would not appear in the corresponding conversion of a CFG into RTN. To this regard, it is remarkable the observation found in [Woods 2010] relating the fact that some parts of a CFG are essentially used to represent what can be obtained with finite state models:

> *With an RTN it is possible to factor together common parts of different context-free rules while maintaining the constituent structure of the original. In an RTN, one can distinguish a finite-state portion of the machinery and a constituent-structure portion of the machinery and keep the two distinct.*

RTNs as a normal form of CFGs

Let us see how a RTN can be considered just a graphical representation of a particular type of CFG which, in our humble opinion, could be considered a normal form.[28]

Let us try to represent RTNs in the form of CFGs by *mimicking* the relationship between right-linear grammars and finite state automata: a transition from q_A to q_B with a label a would correspond to the rule $A \rightarrow aB$. The basic difference between FSA and RTNs are transitions labeled with model references. A naive application of the previous translation rule to such transitions from q_A to q_B using a reference to C would produce rules of the form $A \rightarrow CB$ (being $A, B, C \in N$), as illustrated in Figure 61. This type of rules can also be found in CFGs in the Chomsky normal form. When the destination state is the

26 Dotted rules are defined below. I have always been a little bothered by the fact that grammars and automata *seem* too different due to the use of different concepts and terminology. This question have been even used as the title of a paper [Wirth 1977]: *What can we do about the unnecessary diversity of notation for syntactic definitions?*

27 Which is not the same as minimizing the RTN.

28 We have not found it in the literature, but it is very similar to (a restriction of) 2NF.

special final state q_F, the generated CFG rule would drop this state in the right-hand side (as with FSA), since there is no non-terminal associated to it.

It turns out that the grammar, after this transformation, produces the same language as the following interpretation of the RTN:

- there is a language associated to each non-final state, which corresponds to the set of paths from this state to the final state;
- assuming that transitions are labeled with weighted languages (using the weighted language semiring) and that labels with terminal symbols are interpreted as containing the language generating just this label, it suffices to interpret model reference transitions as labeled with the language generated by its non-terminal (or state) symbol.

That is one of the reasons why we prefer the interpretation of RTNs using state references instead of using a set of independent finite state networks or sub-automata associated to each non-terminal.

bilinear grammars

Let us observe that the presence of rules of the form $A \to BC$ convert these grammars into bilinear. We will see that the cost of some parsing algorithms depend on the rank of the grammar (the maximum length of the right-hand-side of rules) so that the use of binary rules is mandatory, in practice, because they have the minimum rank to represent general CFGs.[29] This can be achieved by converting the grammar to a binary form (usually using the CNF or variants) or by applying an implicit on-the-fly binarization as is the case of the use of dotted-rules in LR parsers and in Earley parsing.

an alternative to CNF and CYK

The use of RTNs as a normal form for CFGs seems an alternative to CNF and the associated algorithms an alternative to CYK (see Section 5.6). The proposed normal form is not CNF and, indeed, is a particular case of the "canonical two form" (2NF) described before. Let us remember that production rules of a 2NF grammar are of the form $A \to \alpha, \alpha \in (\Sigma \cup N)^\star, |\alpha| \leqslant 2$. A careful observation to the set of rules produced from a RTN shows that we are dealing with a restricted version of 2NF where rules of the form $A \to ab$ and $A \to Ba$ (where $A, B \in N, a, b \in \Sigma$) never appears. There is no problem forbidding those rules since they can easily be replaced, for any 2NF grammar, by creating a new non-terminal symbol associated to each terminal symbol appearing at the right-most part of a rank 2 rule.[30]

As pointed out in Section 5.4.1, normal forms require some constructive procedure to convert other CFGs to them. What we seek here is, in essence, to construct RTNs from CFGs. A literature review on the representation of RTNs by means of CFG reveals that most techniques are related to the concept of "bilinear cover" [Leermakers 1989; Section 3]: a grammar G is covered by another grammar G′ if both generate the same language and if it is also possible to retrieve the parse

29 Linear languages are those generated by a linear grammar, which contains at most one non-terminal in the right-hand side of the rules. Some linear context free languages are proven not to be regular, as is the case of the Dyck language (the language of well-balanced bracket pairs).

30 Otherwise stated, we can basically concentrate on how to obtain a 2NF form and replace rules of the form $A \to ab$ by $A \to a\bar{B}$ and $A \to Ba$ by $A \to B\bar{A}$ where $\bar{A} \to a$ and $\bar{B} \to b$ are new auxiliary symbols associated to the terminal ones.

trees associated to G from the parse trees obtained for G′. A bilinear cover is a cover which is also bilinear. [Graham *et al.* 1980; Section 4] describes a method to obtain a cover which is not only bilinear but also 2NF. This method appears with some slightly variations in other works [Leermakers 1989; Nederhof 2000] and consists in associating a new non-terminal to each dotted-rule.

Given grammar G and a rule from G of the form $A \to \alpha\beta$, and assuming that the symbol " $\cdot$ " does not belong to $N \cup \Sigma$, the expression $\langle A \to \alpha \cdot \beta \rangle$ is called a dotted-rule of the grammar. A dotted-rule is just a rule annotated with a position in the right-hand side. Dotted-rules are used in LR and other parsing algorithms as is the case of Earley parsing [Earley 1970] or variations.

By using dotted-rules as new non-terminals, the transition from $\langle A \to \alpha \cdot a\beta \rangle$ to $\langle A \to \alpha a \cdot \beta \rangle$ means that the right-hand side symbol $a \in \Sigma$ has been somewhat processed, which can be described by the following rule of the cover grammar G′ [Graham *et al.* 1980; Section 4]:

$$\langle A \to \alpha a \cdot \beta \rangle \to \langle A \to \alpha \cdot a\beta \rangle \langle a \rangle \tag{5.12}$$

where $\langle a \rangle$ is a new non-terminal of G′. Similar rules are described for analogous cases such as:

$$\langle A \to \alpha B \cdot \beta \rangle \to \langle A \to \alpha \cdot B\beta \rangle \langle B \to \gamma \cdot \rangle \tag{5.13}$$

for every rule $B \to \gamma$.

An alternative definition is found in [Nederhof 2000; Section 4.1]. Although it does not explicitly talk about dotted rules, it is roughly equivalent and consists in distinguishing $m + 1$ new states associated to the same number of positions where a dot may be placed in a production rule $A \to \alpha$ of rank $|\alpha| = m$. Two additional states q_A and $q_{A'}$ are created for each non-terminal A in order to collect the states associated to productions of A into an independent sub-automaton.

simpler approach

We find these techniques quite cumbersome and we advocate for using the much simpler approach of [Lange and Leiß 2009; Section 4.1] where each rule of the form $A \to x_1 x_2 \cdots x_n$, with $n > 2$, is replaced by the rules:

$$\begin{aligned} A &\to x_1 \langle x_2, \ldots, x_n \rangle \\ \langle x_2, \ldots, x_n \rangle &\to x_2 \langle x_3, \ldots, x_n \rangle \\ &\vdots \\ \langle x_{n-1}, x_n \rangle &\to x_{n-1} x_n \end{aligned}$$

Where $\langle x_1, \ldots, x_k \rangle$ represent a new non-terminal *shared by all production rules of the new grammar* G′, meaning that if the same sequence appears in the right-hand side of different productions (even from different non-terminals) the new non-terminal remains the same since. For instance, the following rules:

$$\begin{aligned} A &\to BCD \\ B &\to ECD \end{aligned}$$

would lead to rules which share the new non-terminal $\langle C, D \rangle$. Moreover, the grammar G′ obtained in this way is a left-cover of G, meaning

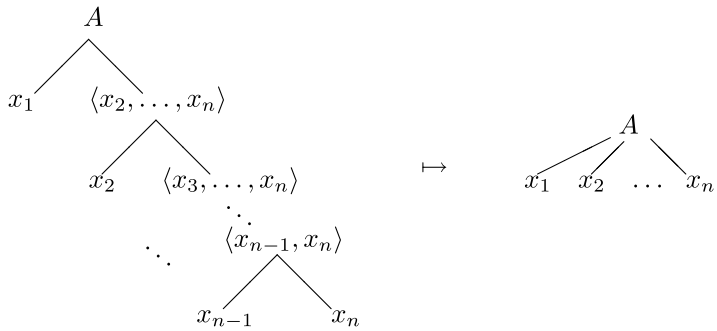


Figure 62: Left cover of a binarized grammar (from [Lange and Leiß 2009; Section 4.1]).

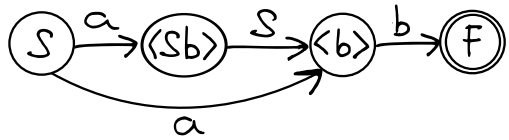


Figure 63: RTN of language $\{a^n b^n \mid n > 0\}$ associated to CFG $S \to aSb|ab$.

that there is a mapping between leftmost derivations of G and leftmost derivations of G′, as illustrated in Figure 62.

Let us see an example of how a CFG can be translated into a RTN by using the following un-weighted[31] grammar associated to the language $\{a^n b^n \mid n > 0\}$.

$$S \to aSb|ab$$

The conversion into RTN-form (a restriction of 2NF where some terminals are replaced by auxiliary non-terminals) is as follows:

$$\begin{aligned} S &\to a\langle Sb\rangle|a\langle b\rangle \\ \langle Sb\rangle &\to S\langle b\rangle \\ \langle b\rangle &\to b \end{aligned}$$

which corresponds to the RTN depicted in Figure 63.

Note that the proposed 2NF transformation has some important advantages w.r.t. the more popular CNF transformation:

- it is simpler;
- the structure of parse trees is preserved; and
- the size of the resulting grammar only grows linearly.

There are other proposals for converting CFGs into 2NF. We have described a right-binarization but, for instance, left-binarization is already described in [Aho and Ullman 1972]. Indeed, the number of possible binarizations of a given rule is related with the possible ways its right-hand side can be parenthesized and this grows with its rank following the series of Catalan numbers. Grammar binarization techniques have an influence on the size of the binarized grammar size

31 The weighted case is straightforward.

(although finding the smallest binarized grammar is NP-hard [Gómez-Rodríguez 2014]) and is also related to parser efficiency [Song *et al.* 2008].

In order to boost the practical importance of 2NF we need parsing algorithms which can directly deal with grammars in this form. Remark that there are variants of CYK admitting 2NF grammars. Let us briefly describe the approach described in [Lange and Leiß 2009]: In a first step, the set of nullable non-terminals of a 2NF grammar G':

$$\mathcal{E}_{G'} = \{A \in N \mid A \xrightarrow{\star} \epsilon\} \tag{5.14}$$

as well as the unit production relation:

$$\mathcal{U}_{G'} = \{(A, y) \in N \times \Sigma \mid \exists \alpha, \beta \in \mathcal{E}^{\star}_{G'}, A \to \alpha y \beta\} \tag{5.15}$$

are precomputed. The transitive closure of this relation $\mathcal{U}^{\star}_{G'}$ is used to tackle the rules of the grammar of the form $A \to \alpha, |\alpha| \leqslant 1$. Unfortunately, the algorithm is described for the non-weighted case and for processing an input string, not a more general word graph. Although it seems easy to generalize this algorithm to the weighted case and to word graphs,[32] we prefer to devote our effort to an alternative algorithm (described in then next section) to compose a weighted FSA with a weighted RTN using a top-down filtering approach more commonly found in Earley parsing or in transducer composition algorithms.

Observe that the pre-computation described before can also be found in some left-to-right top-down parsers, as is the case of the probabilistic Earley parser (as described in [Stolcke 1995]) where the reflexive transitive left-corner relation[33] is extended to the probabilistic case.

Let us conclude this section by remarking that there may be other techniques to obtain or to improve the RTN representation of a CFG. In particular, some techniques related to the regular approximation of CFGs [Nederhof 2000; Mohri *et al.* 2001] and to the notion of strongly regular grammars[34] seems quite promising.

32 The extension of CYK to the weighted case is usually known as inside/outside. Although we have not found a precise description of this algorithm for 2NF grammars, the probabilistic extension of the unit production relation has been defined elsewhere (e.g. [Stolcke 1995; Section 4.5]).

33 The left-corner relation basically computes the set of symbols and reference transitions which can be reached from a given state without consuming symbols (using only model references as *expansions* and empty transitions).

34 A grammar is self-embedding [Chomsky 1959] if there is some $A \in N$ such that there is a derivation from A to $\alpha A \beta$ where $\alpha \neq \epsilon$ and $\beta \neq \epsilon$. Grammars that are not self-embedding are called strongly regular and they generate regular languages [Chomsky 1959; Nederhof 2000]. This is related with the partition of the set of non-terminals into sets of mutually recursive non-terminals. Given the set of non-terminals N of a CFG $G = (N, \Sigma, P, S)$, two non-terminals A and B are *mutually recursive* [Mohri *et al.* 2001] if it is possible to derive from one another (together with other symbols), e.g. if it is possible to derive both $A \xrightarrow{\star} \alpha B \beta$ and $B \xrightarrow{\star} \alpha' B \beta'$ being $\alpha, \beta, \alpha', \beta' \in (N \cup \Sigma)^{\star}$. This leads to a binary relation with the properties of a binary equivalence class producing a partition of N into sets of mutually recursive non-terminals. In strongly regular grammars, each of this partitions is either left-linear or right-linear so that it is clear that the CFG can be converted into an automaton [Mohri *et al.* 2001]. The left-linear models would lead to less efficient RTNs than converting them into right-linear.

5.5 DERIVING THE COMPOSITION OF A REGULAR AND A CONTEXT FREE MODEL

parsing as intersection

We have seen that some quite general cases of parsing can be viewed as the composition of two weighted transducers,[35] which is the generalization, to the weighted case, of the paradigm known as "parsing as intersection". Although these ideas are not at all new, they are not usually proposed as the primary way to introduce parsing algorithms.

There are several constructive procedures for combining weighted languages and transducers from some particular families. In particular, the combination of two CFGs is undecidable [Hopcroft and Ullman 1979], but the combination of a regular and a context free model is context free. The Bar-Hillel algorithm [Bar-Hillel *et al.* 1961] is one of such constructive techniques which has been more recently extended to the weighted case [Nederhof and Satta 2003; Dyer 2010]. This and other classical algorithms are reviewed in Section 5.6, where some of their features will be succinctly compared with the techniques described in this section. It may seem strange to propose an algorithm before describing the most classical ones. Our intention has been to try to explain how a parsing could be *derived* without the influences of terms and concepts commonly found in the literature.[36] Here, we will try to derive it from: 1) the definitions of weighted languages/transducers, 2) from the description of CFGs in terms of language equations, and 3) from the equivalence of description of regular and context free languages and transducers as finite state devices. This may even help to better relate those algorithms afterwards. In particular, it is desirable to relate parsing of regular and context free models as much as possible: Since a FSA seem a particular case of a RTNs, the composition of two FSA is just a particular case of the composition of a FSA with a RTN. An interesting problem we have not taken into account is the replacement of a FSA input by an acyclic RTN without recursion cycles[37] in a more efficient way than the straightforward approach consisting in flattening[38] the model.

This intent is not only aimed at relating the parsing of context free and regular models (which has been clarified only sometimes in the vast literature) or to try to explain them without resorting to charts, dotted rules and the like. It is also motivated by the fact that, in many cases, parsing algorithms are given *as is*, sometimes in the form of a set of (weighted) logical rules[39] which are not derived from the previous

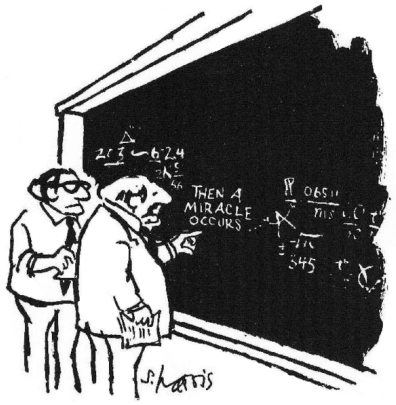
35 Note that we are dealing with weighted transducers as a quite straightforward extension to weighted languages (as in Equation 5.1) or just by replacing languages by transducers in the elemental constituents of grammars, even for CFGs. We have to remark, nevertheless, that CFG transducers as described in this work should not be confused with synchronous CFGs [Chiang 2004].

36 We believe that this will be profitable for readers without a background in the field, although it is improbable that those readers are interested in (and aware of) this work.

37 As mentioned previously, this particular case of RTN is an interesting way to represent packed shared forests.

38 Replace the model references by their definition making a copy renaming states. This is similar to the approach to decoding by expansion described in Section 5.5.3 but, in this case, is a finite procedure which could be performed beforehand.

39 Particularly in the case of "semiring parsing" [Goodman 1999] where the use of weighted logic deduction is, unfortunately, implicit in the term.



"I think you should be more explicit here in step two."

Figure 64: Some works introduce the formal description of weighted languages and transducers just to propose parsing and decoding algorithms not directly derived from the former definitions. This is compensated, to a certain extent, by informal explanations and, in some cases, by formal proofs of the correctness. Cartoon reprinted with permission by S. Harris. Copyright by `ScienceCartoonsPlus.com`.

definitions. This reminds us a famous cartoon (see Figure 64) where, in some part of a mathematical demonstration "THEN A MIRACLE OCCURS ...". In the case of parsing, the lack of a clear derivation is compensated, to a certain extent, by informal intuitions and, in some cases, by a proof of correctness of the proposed algorithms.

5.5.1 State-pair transducer composition

We have seen so far that weighted transducers may be organized in a semiring algebraic structure in at least two different ways (using either transducer composition or language concatenation as the product). In the following, the operator $\otimes$ will denote language concatenation[40] whereas the use of composition will be denoted, as usual, with $\circ$.

The first model of the composition described in this section is a right-linear grammar G_I representing a transducer from $\Sigma^\star \times \Omega^\star \to \mathbb{K}$ and is intended to be the input of the parsing process. The grammar G_M is the second transducer of the composition and represents a RTN transducer from $\Omega^\star \times \Phi^\star \to \mathbb{K}$. We seek to obtain the composition[41] $G_O = G_I \circ G_M : \Sigma^\star \times \Phi^\star \to \mathbb{K}$.

The rules of G_I are of the form $A \to wB$ or $A \to w$, where w can be expressed as $\{(x : y) \cdot k\})$ (with $x \in \Sigma^\star, |x| \leqslant 1$, $y \in \Omega^\star, |y| \leqslant 1$). The use of pairs $x : y$ is a natural extension from languages to transducers, whereas k (the weight associated to the rule) can be omitted, for succinctness, in some descriptions.

We have also seen that a regular model in the form of a right-linear grammar can be represented with a FSA whose edges are labeled with values from a language semiring and whose states q_A, excepting the final state q_F, are associated to the corresponding non-terminal A of the grammar. That is, G_I can be represented by means of a FSA where $Q = \{q_A : A \in (N \cup \{F\})\}$ is the set of states, N is the set of non-terminals of G_I, and where $F \notin N$ is a final state as described in Section 5.4.2.

40 And concatenation of transducers, as defined in Equation 5.8.

41 Sub-indices I, M and O denote, respectively, the terms *Input*, *Model* and *Output*.

Observe that, in this case, a transition from q_i to q_j, denoted by $tr(q_i, q_j)$, may correspond to several rules of the grammar and are sometimes graphically depicted with several edges.[42]

The language associated to each non-terminal A of G_I corresponds to the sum of values of paths from q_A to q_F in the FSA. Since the sum of values associated to all the paths[43] of a given span from q to q' will be used many times, it will be denoted as: $[q, q']$. This value is also known as the distance or shortest distance between q and q' since it coincides with this the classical definition of shortest distance using certain types of semiring.[44] Using this notation, the relationship between grammar non-terminals and automaton states can be described in this succinct way: $A = [q_A, q_F]$. This value can be formally expressed as:

shortest distance

$$[q, q'] = \bigoplus_{(e_1, \dots e_n) \in p_e(q, q')} \bigoplus_{i=1}^{n} e_i \tag{5.16}$$

where $p_e(q, q')$ denote the set of paths from q to q' represented by sequences of *edges*. Another way to represent the value of the span consist in grouping the values of each transition as follows:

$$[q, q'] = \bigoplus_{(q_1, \dots q_n) \in p_{st}(q, q')} \bigoplus_{i=1}^{n-1} \left(\bigoplus_{e \in tr(q_i, q_{i+1})} e \right) \tag{5.17}$$

where $p_{st}(q, q')$ is the set of paths defined as *state* sequences. If we consider that a transition $tr(q, q')$ is represented by a value grouping the corresponding set of edges, we can also represent the former equation as follows:

$$[q, q'] = \bigoplus_{(q_1, \dots q_n) \in p_{st}(s, t)} \bigoplus_{i=1}^{n-1} tr(q_i, q_{i+1}) \tag{5.18}$$

This value can also be described from the transitive closure (denoted by the $\star$ operator) of the transition matrix of the automaton, which can be expressed as an infinite sum of matrices associated to paths of different lengths. It is also possible to define $[q, q']$ by means of a system of equations from the following expansion:

$$[q, q'] = \bigoplus_{q_{aux} \in Q} tr(q, q_{aux}) \otimes [q_{aux}, q'] \tag{5.19}$$

Observe that this leads to the same set of linear equations as the right-linear grammar of Section 5.4.2 by taking into account the fact that there are not transitions departing from q_F and that $[q_F, q_F] = \{\epsilon : \epsilon\} \cdot 1$.

42 There may be several edges because we can have several rules $A \to a_1 B$, $A \to a_2 B$, $\dots$, $A \to a_n B$. Representing them with just one edge (with the corresponding language semiring value) is a matter of taste since we can either consider a non-simple graph with several transitions or a simple graph labeled with sets of transitions (a graph is *simple* when there is at most one edge from a source to a destination).

43 We are assuming a complete semiring, since the set of paths may be not finite. Indeed, the unary clausure $\star$ operation is also used to describe this value.

44 Note that the definition of shortest distance makes the summation of all paths into account. This value coincides with the conventional concept of shortest distance when using the Viterbi, the artic or the tropical semirings. The semiring used to label the edges of the FSA (viewed as a graph) should not be confused with the semiring $\mathbb{K}$ associated to the weighted languages and transducers. The later one is used, in our case, to label FSA transitions.

ignored for the regular case

The spans $[q, q']$ constitute the variables of the equations, but observe that the final state of those spans is always q_F if the original span to be expanded in Equation 5.19 is $[q_S, q_F]$. That is why the second state of the span notation is usually ignored in works associating a language to each automaton state. However, this second state will become relevant, for RTNs, when composing a FSA with a RTN.

In a similar way, the rules of G_M are given in the RTN-form described in previous section: a restriction of the 2NF normal form which can also be viewed as an extension of right-linear grammars with the inclusion of rules of the form $A \to BC$. This grammar can also be represented by means of a finite state device, although we will use letter r to denote their states in order to distinguish them from the states of G_I, denoted by q. This finite state RTN is very similar to the FSA excepting the case of transitions associated to bilinear rules $A \to BC$ which are represented by transitions from r_A to r_C labeled with a reference to the language associated to B, represented by the span $[r_B, r_F]$. Since these spans always finish with r_F, we can simply depict them, in an abuse of notation, with its first state (r_B in the example).

In order to formally specify the composition $G_I \circ G_M$, we will start by replacing G_I and G_M by their respective languages represented in terms of the spans of their finite state representation:

$$G_I \circ G_M = [q_S, q_F] \circ [r_S, r_F] \tag{5.20}$$

where q_S and r_S are the states associated to the axioms of grammars G_I and G_M, respectively.[45]

proposed constructive strategy

We propose a constructive strategy to obtain an algorithm for combining these spans: our aim is to obtain a system of equations of the composed transducer by manipulating the descriptions of these spans. As will be shown, distribution and factorization of composition and semiring operations, in these descriptions, is limited to special cases.

Given the transducers $f, f' : \Sigma^\star \times \Omega^\star \to \mathbb{K}$ and $h, h' : \Omega^\star \times \Phi^\star \to \mathbb{K}$, it is easy to check that composition can be distributed over summation:

$$(f \oplus f') \circ h = (f \circ h) \oplus (f' \circ h) \tag{5.21}$$

$$f \circ (h \oplus h') = (f \circ h) \oplus (f \circ h') \tag{5.22}$$

Unfortunately, language concatenation cannot be distributed with composition in the general case.[46] Nevertheless, there are some special or particular cases where it is possible. For instance, the following equations hold when the semiring $\mathbb{K}$ is commutative:

- $(\{w:\epsilon\} \otimes f) \circ h = \{w:\epsilon\} \otimes (f \circ h)$, being $w \in \Sigma^\star$; (5.23)
- $f \circ (\{\epsilon:w'\} \otimes h) = \{\epsilon:w'\} \otimes (f \circ h)$, where $w' \in \Phi^\star$; and (5.24)
- $(\{w:a\} \otimes f) \circ (\{b:w'\} \otimes h) = \begin{cases} \{w:w'\} \otimes (f \circ h), & \text{if } a = b \\ \emptyset & \text{otherwise} \end{cases}$ (5.25)

  being $w \in \Sigma^\star, a, b \in \Omega, w' \in \Phi^\star$.

45 That is, S and F are used to denote, respectively, the axiom and the artificial final state for both models since the state (q or r) avoids ambiguities.

46 There is no general way to simplify $(f \otimes g) \circ h$.

Note that languages $\{w:\epsilon\}$, $\{\epsilon:w'\}$, $\{w:a\}$ and $\{b:w'\}$ from the above equations should be understood, each one, as accompanied by a semiring value $k \in \mathbb{K}$ (e.g. $\{(w:\epsilon)\cdot k\}$ and so on) which is not displayed for succinctness. Note also that the second and the third equations do not correspond to Equation 5.2 when using a non-commutative semiring since the weights associated to each model of the composition are not interleaved.[47] This remark is relevant for the classical transducer composition based on state pairs described below. From now on, unless otherwise stated, we will only work with commutative semirings.

commutative semirings

Note also that we have to be cautious since some apparently correct expressions do not always hold. For instance, $[q,q]$ is not necessarily $\{\epsilon:\epsilon\}\cdot 1$ since there may exist loops.[48] Also, it may not be obvious to determine when the following expression holds:

$$[q_i,q_j] \stackrel{?}{=} \bigoplus_{q_k\in Q} [q_i,q_k]\otimes[q_k,q_j] \tag{5.26}$$

where the sum of all paths from q_i to q_j is described as the sum of paths using an auxiliary state q_k. This expression is not correct for non-idempotent semirings because many paths are counted several times as illustrated in Figure 65.

Figure 65: The path from q_i to q_j using q_k and the path from q_i to q_j using $q_{k'}$ are the same. In this way, some paths are counted several times in Equation 5.26, which may not pose problems in some particular semirings but it is incorrect in the general case (not necessarily idempotent semirings).

The recursive[49] description of the span associated to a RTN is more complex because of the transitions labeled with other spans. The spans labeling transitions are always finished by the final state r_F.

Now, in order to compute $G_I \circ G_M$, let us try to express an expression of the form $[q_i,q_j]\circ[r_x,r_y]$ using the recursive descriptions, which involves expansion of expressions.

Let us start by dealing with transitions labeled with a symbol in the common tape Ω, leaving aside, for the moment, transitions with model references and transitions with the null string in the common tape. This corresponds to the classical composition of finite state transducers. In this case, it is possible to expand $[q_i,q_j]\circ[r_x,r_y]$ using Equations 5.21 and 5.22 to obtain an expression with the following structure:

$$\left(\bigoplus_{q_k\in Q} tr(q_i,q_k)\otimes[q_k,q_j]\right)\circ\left(\bigoplus_{r_z\in R} tr(r_x,r_z)\otimes[r_z,r_y]\right) \tag{5.27}$$

47 Strangely, we have not found this observation in the literature. The use of non-commutative semirings poses also problems in the equivalence between inside-outside and forward-backward.

48 However, shortest distance problems for certain semiring types may assume that. Note also that, for the particular case of $[q_F,q_F]$ we can assume that it is equal to $\{\epsilon:\epsilon\}\cdot 1$ since there are no outgoing transitions.

49 A recursion with cyclic dependencies, as explained before, since this leads to a (now, not necessarily linear) system of language equations.

which can be converted, applying Equations 5.21 and 5.21, into:

$$\bigoplus_{q_k \in Q, r_z \in R} (tr(q_i, q_k) \otimes [q_k, q_j]) \circ (tr(r_x, r_z) \otimes [r_z, r_y]) \tag{5.28}$$

Note that each transition $tr(\cdot,\cdot)$ can also be interpreted as a summation of zero, one or a finite number of values (including model references to languages/transducers, denoted by $[r_A, r_F]$, in the case of RTNs). The result of this expansion is a finite set of addends where each one has the following form:

$$\begin{aligned}(\{(a:b)\cdot k_1\} \otimes [q_k, q_j]) \circ (\{(c:d)\cdot k_2\} \otimes [r_z, r_y]) = \\ = \begin{cases} \{(a:d)\cdot (k_1 \otimes k_2)\} \otimes ([q_k, q_j] \circ [r_z, r_y]), & \text{if } b = c \\ \emptyset, & \text{otherwise} \end{cases}\end{aligned} \tag{5.29}$$

where $b, c \in \Omega$, $a \in \{\epsilon\} \cup \Sigma$, and $d \in \{\epsilon\} \cup \Phi$. As can be observed, components of transitions can either match or not. Therefore, we can easily simplify the summations of Equation 5.28 as follows:

$$\begin{aligned}(tr(q_i, q_k) \otimes [q_k, q_j]) \circ (tr(r_x, r_z) \otimes [r_z, r_y]) = \\ = (tr(q_i, q_k) \circ tr(r_x, r_z)) \otimes ([q_k, q_j] \circ [r_z, r_y])\end{aligned} \tag{5.30}$$

Assuming the convention (used in the interpretation of GFGs as systems of equations) that a summation of the form $X = \bigoplus_{i \in I} Y_i$ can be represented by means of a set of rules $X \to Y_i$, the expression of Equation 5.27 could be represented by using rules of the form:

$$([q_i, q_j] \circ [r_x, r_y]) \to \{(a:b)\cdot k_1\} \otimes ([q_k, q_j] \circ [r_z, r_y]) \tag{5.31}$$

where $a \in \{\epsilon\} \cup \Sigma$, $b \in \{\epsilon\} \cup \Phi$. Now, the idea is to identify the terms $[q_i, q_j] \circ [r_x, r_y]$ as variables of a system of equations. Since we are trying to compute the initial expression $[q_S, q_F] \circ [r_S, r_F]$ and the second state of each span is not modified, the expanded expressions are of the type $[q, q_F] \circ [r, r_F]$ which can be denoted as $\langle q, r\rangle$. In this way, the rules of Equation 5.31 can be rewritten as:

$$\langle q, r\rangle \to \{(a:b)\cdot k_1\} \otimes \langle q', r'\rangle \tag{5.32}$$

which can be interpreted as rules of the form $A \to aB$ constituting a right-linear CFG. The rules of the form $A \to a$, required to have a useful grammar, are obtained by removing $\langle q', r'\rangle$ from the right-hand side when q' and r' become q_F and r_F, respectively.[50] Indeed, $\langle q_F, r_F\rangle$ does not constitute a non-terminal of the grammar but it is interpreted as the special auxiliary state F used to construct a FSA from a right-linear CFG. The elements $\langle q, r\rangle$ can also be viewed as states of a FSA. The transitions $tr(\langle q, r\rangle, \langle q', r'\rangle)$ can be computed as follows:

$$tr(\langle q, r\rangle, \langle q', r'\rangle) = \left\{(a:c)\cdot k \mid a \in \Sigma, c \in \Phi, k = \bigoplus_{\substack{b \in \Omega \\ (a:b)\cdot k_1 \in tr(q,q') \\ (b:c)\cdot k_2 \in tr(r,r')}} k_1 \otimes k_2\right\} \tag{5.33}$$

50 Note that, when only one of them is q_F or r_F, the resulting state is not useful.

Data: $G_I : \Sigma^\star \times \Omega^\star \to \mathbb{K}$ a FSA without ϵ transitions,
$G_M : \Omega^\star \times \Phi^\star \to \mathbb{K}$ a FSA without ϵ transitions
Result: $G_O = G_I \circ G_M : \Sigma^\star \times \Phi^\star \to \mathbb{K}$ a FSA without ϵ transitions
$Q' \leftarrow \{\langle q_S, r_S\rangle, \langle q_F, r_F\rangle\}$; $tr' \leftarrow \emptyset$; $S \leftarrow$ queue($\{\langle q_S, r_S\rangle\}$)
while not *empty(S)* **do**
  $\langle q, r\rangle \leftarrow$ extract(S)
  foreach $q \xrightarrow{\{a:b\cdot k_1\}} q'$ **do**
    foreach $r \xrightarrow{\{c:d\cdot k_2\}} r'$ **do**
      if b=c **then**
        if $\langle q', r'\rangle \notin Q'$ **then**
          $Q' \leftarrow Q' \cup \{\langle q', r'\rangle\}$
          enqueue(S,$\langle q', r'\rangle$)
        $tr' \leftarrow tr' \cup \{\langle q, r\rangle \xrightarrow{\{a:d\cdot(k_1 \otimes k_2)\}} \langle q', r'\rangle\}$
return G_O given by Q' and tr'

Algorithm 1: Composition of two FSA without null transitions.

This association of pairs of states to construct a new transducer is the underlying idea of classical transducer composition algorithms for commutative semirings inspired by the FSA intersection algorithm [Hopcroft and Ullman 1979]. Algorithm 1 starts from $\langle q_S, r_S\rangle$ to expand, for each visited state $\langle q, r\rangle$, the Equation 5.27 by combining the outgoing transitions of q and r as described in Equation 5.33. It suffices to use a queue[51] to insert the destination states yet to be examined so that we can to explore the reachable part of the graph.

The traditional rationale to obtain this algorithm is based on the idea of matching paths from each model instead of viewing the models as system of equations of CFGs. We have just provided an alternative way to propose a well known algorithm from the properties of weighted transducers (in a set theoretic sense) w.r.t. the composition and the semiring operations. When we depart from two finite state transducers,[52] a system of equations is obtained which turns out to be a right-linear CFG which can also be represented as a finite state transducer.

5.5.2 Extension to null-transitions

The derivation of the transducer composition algorithm described before cannot be applied *as is* to models with some transitions labeled with the null string ϵ in the common tape which, in an abuse of notation, will be termed null transitions.[53]

51 Any type of queue policy or discipline (LIFO, FIFO,...) is acceptable.

52 The second is aimed to be a RTN, but the restrictions made so far makes it a FST. One of our main aims is to describe the parsing of CFGs as a direct extension of regular models.

53 E.g. a label of the form $a:\epsilon$ from G_I or a label of the form $\epsilon:b$ from G_M. In other contexts, the term "null transition" is restricted to $\epsilon:\epsilon$ or, contrarily, extended to the presence of ϵ in any tape (i.e. also in the tapes associated to Σ and Φ). A drawback of our interpretation of *null transition* is that it is dependent on the position of the model in

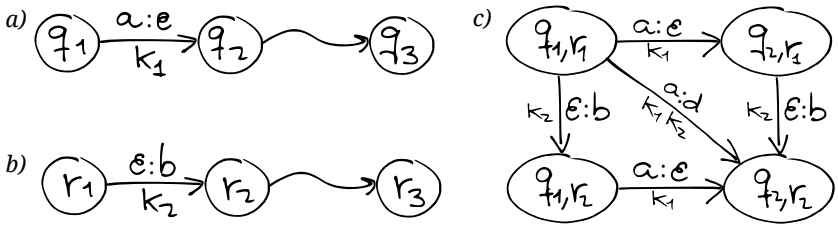


Figure 66: Example of null transitions.

Note that the question is not the correctness of the algorithm when applied to those cases since *the steps followed to derive the algorithm are not even applicable when hypothesizing these other types of transition* and have to be extended. Let us see how to modify the previous cases to deal with null-transitions. As we will see, the trivial application of the classical and well known state-pair approach is not equivalent to the proposed extension.

alternative to filter composition

The solution proposed in the literature to solve the problems of the classical state-pair approach is known as filter composition [Pereira and Riley 1997] and we will compare our proposal with this filter-composition.

The case of transitions with null labels in the common tape associated to the Ω alphabet seems related to Equations 5.23 and 5.24. Let us see different examples to illustrate the problems and to try to provide some insight on the proposed solution.

In a first example, we want to compute $[q_1, q_3] \circ [r_1, r_3]$. There is a transition from q_1 to q_2 labeled with $\{(a{:}b) \cdot k_1\}$ and a transition from r_1 to r_2 labeled with $\{(a{:}b) \cdot k_1\}$, respectively, as illustrated in Figures 66 *a)* and *b)*. It is possible, in this case, to combine the two transitions as follows:

$$(\{(a{:}\epsilon)k_1\} \otimes [q_2, q_3]) \circ (\{(\epsilon{:}b)k_2\} \otimes [r_2, r_3]) = \{(a{:}b) \cdot (k_1 \otimes k_2)\} \otimes ([q_2, q_3] \circ [r_2, r_3]) \quad (5.34)$$

The problem is that, using Equation 5.23, we can also convert $[q_1, q_3] \circ [r_1, r_3]$ into:

$$(\{(a{:}\epsilon)k_1\} \otimes [q_2, q_3]) \circ [r_1, r_3] = \{(a{:}\epsilon)k_1\} \otimes ([q_2, q_3] \circ [r_1, r_3]) \quad (5.35)$$

or, alternatively, using Equation 5.24 we can obtain:

$$[q_1, q_3] \circ (\{(\epsilon{:}b)k_2\} \otimes [r_2, r_3]) = \{(\epsilon{:}b)k_2\} \otimes ([q_1, q_3] \circ [r_2, r_3]) \quad (5.36)$$

However, it is not correct to include the three results as rules of a grammar since the interpretation of the set of rules of a CFG as a system of equations means that the values associated to each non-terminal are included in a summation. Otherwise stated, if we try to expand $[q_1, q_3] \circ [r_1, r_3]$ as in Equation 5.27, we can either apply Equation 5.35, 5.36 or 5.36, *but not more than one at the same time* since this summation is obtained by replacing the terms of Equation 5.19 and several incompatible replacements cannot be performed simultaneously.

the composition. The same transition may be either null or non-null depending of the model where it belongs to (i.e. to G_I or to G_M).

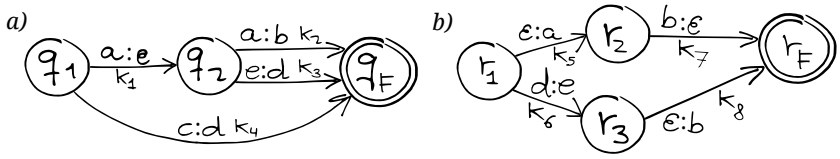


Figure 67: Example of transducers with null transitions: In *a)* the model G_I represents $\{(aa:b)k_1k_2, (ae:d)k_1k_3, (c:d)k_4\}$, the model in *b)*, G_M, is associated to $\{(b:a)k_5k_7, (d:eb)k_6k_8\}$. The composition $G_I \circ G_M$ leads to $\{(aa:a)k_1k_2k_5k_7, (ae:eb)k_1k_3k_6k_8, (c:eb)k_4k_6k_8\}$.

This is the main problem when using the state-pairing approach without modification, as illustrated in Figure 66 *c)* where the value $k_1 \otimes k_3$ is taken into account three times instead of just once.

Faced with this issue, the first idea which might come to our mind is to choose one of the possible expansions. This is a drawback when there are two different types of transitions (with and without ϵ in the common tape, or null and non-null) outgoing from a given state, as in the example depicted in Figure 67.

Let us see what happens when trying to expand $[q_1, q_F] \circ [r1, r_F]$ from the models of Figure 67:

$$(\{\{(a:\epsilon)\cdot k_1\}\otimes[q_2,q_F]\}\oplus\{\{(c:d)\cdot k_4\}\otimes[q_F,q_F]\})\circ \\ (\{\{(\epsilon:a)\cdot k_5\}\otimes[r_2,r_F]\}\oplus\{\{(d:e)\cdot k_6\}\otimes[r_3,r_F]\}) \quad (5.37)$$

There is no problem with:

$$\{\{(a:\epsilon)\cdot k_1\}\otimes[q_2,q_F]\}\circ\{\{(\epsilon:a)\cdot k_5\}\otimes[r_2,r_F]\} = \\ \{(a:a)\cdot k_1k_5\}\otimes([q_2,q_F]\circ[r_2,r_F]) \quad (5.38)$$

or with

$$\{\{(c:d)\cdot k_4\}\otimes[q_F,q_F]\}\circ\{\{(d:e)\cdot k_6\}\otimes[r_3,r_F]\} = \\ \{(c:e)\cdot k_4k_6\}\otimes([q_F,q_F]\circ[r_2,r_F]) \quad (5.39)$$

But it is not easy to try to simplify:

$$\{\{(a:\epsilon)\cdot k_1\}\otimes[q_2,q_F]\}\circ\{\{(d:e)\cdot k_6\}\otimes[r_3,r_F]\} \quad (5.40)$$

which, in the best case, can be converted (Equation 5.23) into:

$$\{(a:\epsilon)\cdot k_1\}\otimes([q_2,q_F]\circ\{\{(d:e)\cdot k_6\}\otimes[r_3,r_F]\}) \quad (5.41)$$

The same happens (Equation 5.24) with:

$$\{\{(c:d)\cdot k_4\}\otimes[q_F,q_F]\}\circ\{\{(\epsilon:a)\cdot k_5\}\otimes[r_2,r_F]\} = \\ \{(\epsilon:a)\cdot k_5\}\otimes(\{\{(c:d)\cdot k_4\}\otimes[q_F,q_F]\}\circ[r_2,r_F]) \quad (5.42)$$

Unfortunately, some terms such as $[q_2, q_F] \circ \{\{(d:e)\cdot k_6\}\otimes[r_3, r_F]\}$ cannot be further simplified to apply the state-pairing approach in a straightforward way.[54] The same problem appears when we try to expand only the transitions outgoing from q_1 or the transitions outgoing from r_1.

54 It would be possible to consider $\{(d:e)\cdot k_6\}\otimes[r_3, r_F]$ as an artificial state of G_M or, similarly, $\{(c:d)\cdot k_4\}\otimes[q_F, q_F]$ as an artificial state of G_I, but we will provide an even better solution.

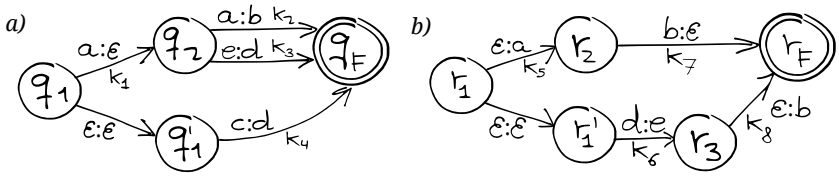


Figure 68: The transducers from the example of Figure 67 after the proposed transformation which makes all transitions outgoing each state as either containing or not containing the null string in the common tape.

There are several generic cases when expanding $[q_1, q_F] \circ [r1, r_F]$:

1. there are not null transitions outgoing from q_1 and there are not null transitions outgoing from r_1: we should expand both of them;
2. all transitions outgoing from q_1 and from r_1 are null transitions: we should also expand both of them;
3. all transitions outgoing from q_1 are null transitions and all transitions outgoing from r_1 are not null transitions: we should only expand q_1 and factorize their transitions;
4. in a symmetric way, all transitions outgoing r_1 are null transitions but any of the transitions outgoing q_1 are null transitions: expand r_1 but not q_1;
5. any combination of expanding or not q_1 and r_1 has some drawbacks when both null and non-null transitions originate from the same state either in G_I or in G_M.

From these observations, it is clear the rationale of our approach:

> *to avoid the existence of states containing both null and non-null outgoing transitions*

How could this feature be achieved? By grouping all non-null transitions in a new auxiliary state which can be accessed from the original state by means of a transition labeled with $\{\epsilon : \epsilon \cdot 1\}$, which is the unit in the weighted transducer semiring. We will call this property "homogeneous epsilon form" and the algorithm to perform this transformation (Algorithm 6) will be discussed into more detail in Section 5.5.4. An example of such transformation, for the models of Figure 67, is depicted in Figure 68. We can easily observe, in this example, that this transformation preserves the resulting weighted transducer.

homogeneous epsilon form

Algorithm 2 is an extension of the former state-pairing approach. It is also based on a queue to perform a traversal of the composed model which is created during this traversal. The algorithm requires that the FSA used in the composition are first converted into homogeneous epsilon normal form, an issue tackled later in Section 5.5.4. There are four different cases depending on the presence of outgoing transitions without the null symbol in the common tape. Although it is possible to construct an on-the-fly implementation of the algorithm, the main restriction is the previous transformation of FSA into the mentioned normal form.

Data: $G_I : \Sigma^\star \times \Omega^\star \to \mathbb{K}$ a FSA in homogeneous epsilon form,
$G_M : \Omega^\star \times \Phi^\star \to \mathbb{K}$ a FSA in homogeneous epsilon form
Result: $G_O = G_I \circ G_M : \Sigma^\star \times \Phi^\star \to \mathbb{K}$ a FSA
$Q' \leftarrow \{\langle q_S, r_S\rangle, \langle q_F, r_F\rangle\}$
$tr' \leftarrow \emptyset$
$S \leftarrow$ queue($\{\langle q_S, r_S\rangle\}$)
def *add_transition(origin* $\xrightarrow{label}$ *dest)*
  if *dest* $\notin Q'$ **then**
    $Q' \leftarrow Q' \cup \{$ dest $\}$
    enqueue(S,dest)
  $tr' \leftarrow tr' \cup \{$origin $\xrightarrow{\text{label}}$ dest$\}$
while not *empty(S)* **do**
  $\langle q, r\rangle \leftarrow$ extract(S)
  if $\exists$ *transition outgoing* q *with no* ϵ *in 2nd tape* **then**
    if $\exists$ *transition outgoing* r *with no* ϵ *in 1st tape* **then**
      `/* this case is similar to Algorithm 1 */`
      foreach $q \xrightarrow{\{x:y\cdot k_1\}} q'$ **do**
        foreach $r \xrightarrow{\{v:w\cdot k_2\}} r'$ **do**
          if $y{=}v$ **then**
            add_transition($\langle q, r\rangle \xrightarrow{\{x:w\cdot k_1\otimes k_2\}} \langle q', r'\rangle$)
    else `/* use only transitions outgoing r */`
      foreach $r \xrightarrow{\{\epsilon:w\cdot k_2\}} r'$ **do**
        add_transition($\langle q, r\rangle \xrightarrow{\{\epsilon:w\cdot k_2\}} \langle q, r'\rangle$)
  else `/* transitions outgoing q have ϵ in 2nd tape */`
    if $\exists$ *transition outgoing* r *with no* ϵ *in 1st tape* **then**
      `/* use only transitions outgoing q */`
      foreach $q \xrightarrow{\{x:\epsilon\cdot k_1\}} r'$ **do**
        add_transition($\langle q, r\rangle \xrightarrow{\{x:\epsilon\cdot k_1\}} \langle q', r\rangle$)
    else `/* use both null transitions simultaneously */`
      foreach $q \xrightarrow{\{x:\epsilon\cdot k_1\}} q'$ **do**
        foreach $r \xrightarrow{\{\epsilon:w\cdot k_2\}} r'$ **do**
          add_transition($\langle q, r\rangle \xrightarrow{\{x:w\cdot k_1\otimes k_2\}} \langle q', r'\rangle$)
return G_O given by Q' and tr'

Algorithm 2: Composition of FSA with null transitions.

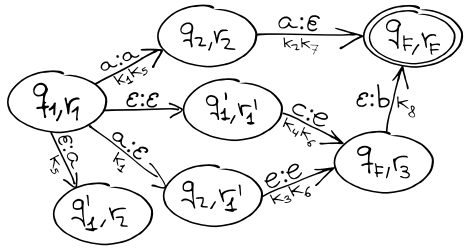


Figure 69: Composition of the models shown in Figure 68. Observe that state $\langle q_1', r_2\rangle$ is not useful. This weighted transducer represents $[\langle q_1, r_1\rangle, \langle q_F, r_F\rangle] = \{(aa{:}a)k_1k_2k_5k_7, (ae{:}eb)k_1k_3k_6k_8, (c{:}eb)k_4k_6k_8\}$, coinciding with $G_I \circ G_M$.

Using this recipe, we can obtain the model of Figures 69, 70, 71 and 72. Observe in this model that there may be useful states where just one of the states of the pairing is the final state.

Let us now compare the proposed solution w.r.t. the technique considered the standard in the state of the art. It is clear that the trivial application of the technique based on pairing states (as illustrated in Figures 66*c* and 70) may lead to incorrect transducers (non-equivalent to the intended composition) unless using idempotent semirings. The number of redundant paths may rapidly grow as illustrated in Figure 70. This problem has been probably first revealed by [Pereira and Riley 1997] and is explained into more detail in [Mohri 2009; Section 5.1]. Their approach to remove all paths excepting one consists in modifying the original models by replacing the ϵ labels of each transducer with new terminal symbols. Two new terminal symbols are used: ϵ_1 is used to replace the original ϵ of the common tape in G_I to obtain $\tilde{G}_I$. Another symbol, ϵ_2, is used to label with $\epsilon{:}\epsilon_2$ an artificial loop placed at each state of Q. The same symbols are used, in a reversed way, in the second transducer G_M to obtain $\tilde{G}_M$.[55] In this way, it is possible to specify, by means of an intermediate transducer, which combination of transitions are allowed,[56] hence limiting the paths to one. Since this intermediate transducer can be specified by means of a finite state model F, known as *filter transducer* or *composition filter*, it suffices to compose the modified models with it:

filter composition

$$G_I \circ G_M = \tilde{G}_I \circ F \circ \tilde{G}_M \tag{5.43}$$

Observe that the composition no longer needs to take into account the presence of null symbols in the common tape. Moreover, the states of the resulting model are no longer pairs but 3-tuples of states since we have to specify the composition with the filter transducer.

There are several filter transducer possibilities which can be tailored for specific purposes or to improve the efficiency of the resulting model [Allauzen *et al.* 2011]. For instance, the removal of redundant paths can be obtained by means of the epsilon-matching or by means of the epsilon-sequencing filters. Note that this increase in the number of states is limited by a small constant factor since the number of states

55 ϵ_2 replaces ϵ and artificial loops with $\epsilon_1{:}\epsilon$ are added at each state of R.

56 For instance, combination of artificial loops are disallowed and directions (diagonal, horizontal, vertical) are prioritized.

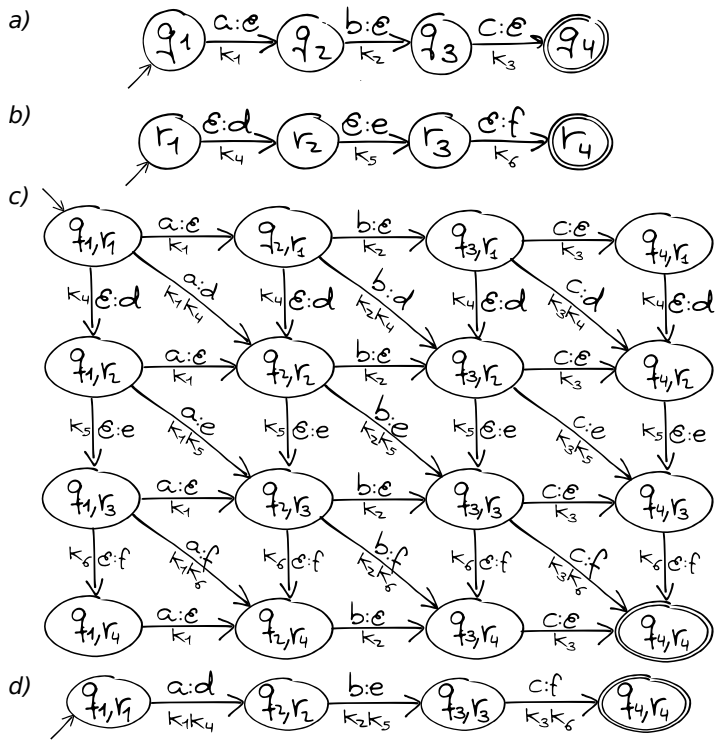


Figure 70: The naive transducer composition algorithm leads to redundant paths which are not equivalent to the intended composition value unless using idempotent semirings: a) and b) are transducers, c) is their naive composition based on pairing states and c) is the result of applying Algorithm 2. Observe that there are several paths in c) from $\langle q_1, r_1 \rangle$ to $\langle q_4, r_4 \rangle$, each path has the value $k_1 \cdot k_2 \cdot k_3 \cdot k_4 \cdot k_5 \cdot k_6$ but only one of them should be taken into account.

of these filters is quite low. Moreover, the modified transducers $\tilde{G}_I$ and $\tilde{G}_M$ do not need to be explicitly constructed since the algorithm can be tailored to directly perform this composition. Note also that this approach is a way to explain that we have to remember which transitions have been actually composed in order to avoid redundant paths[57] and to be able to express them using the state pairing approach. The filter composition approach can be considered ingenious but, in our humble opinion, our technique seems simpler and is motivated by a constructive approach. We are glad to propose something new, to the best of our knowledge, in a quite exploited niche.

The problem with null transitions can also appear as a particular instance of the use of model reference transitions since, in this case, the language associated to the auxiliary symbols of the CFG, represented by means of G_M, may contain some pairs of sentences with the empty word in the common tape . This is the problem of nullable auxiliars (or non-terminal) in the classical parsing literature. Let us first try to deal with model reference transitions without worrying about that in order to later extend the approach to the more general case.

57 Note that some forbidden transition combinations can be used as part of other paths. The state of the filter transducer remembers which paths can be combined or not.

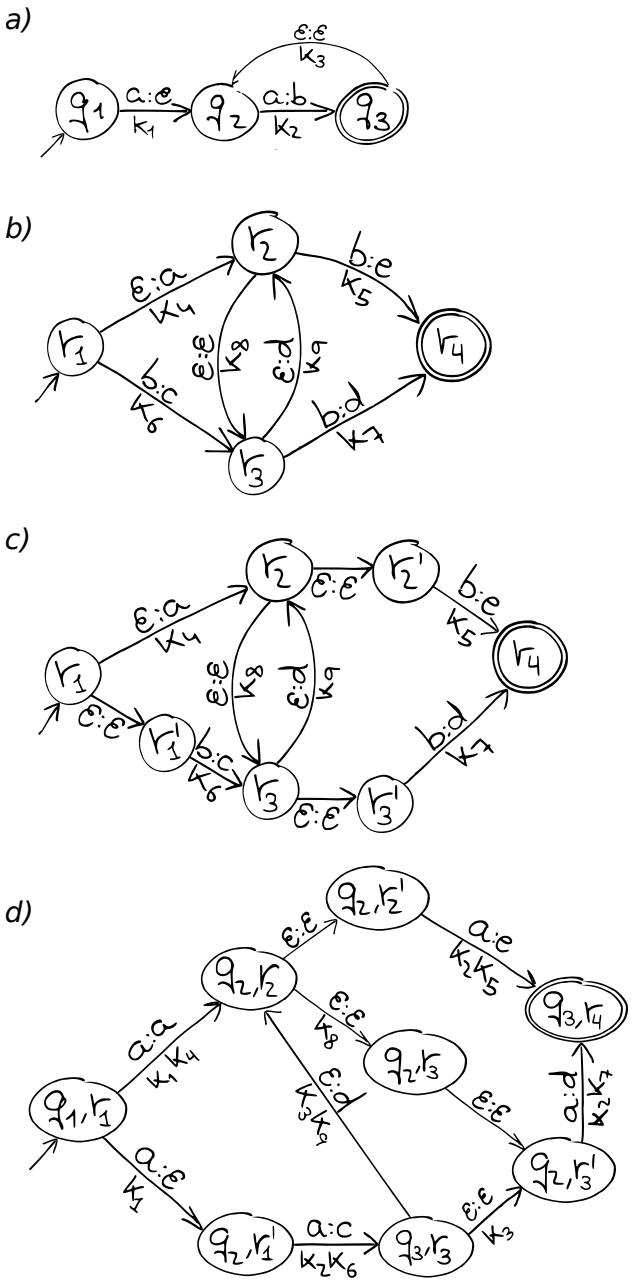


Figure 71: Example of composition of two FSA with epsilon transitions: a) and b) are two weighted FSA, c) corresponds to b) converted into homogeneous epsilon form (a) already fulfills the conditions) and, finally, d) is the result of composing the models from a) and c) using Algorithm 2.

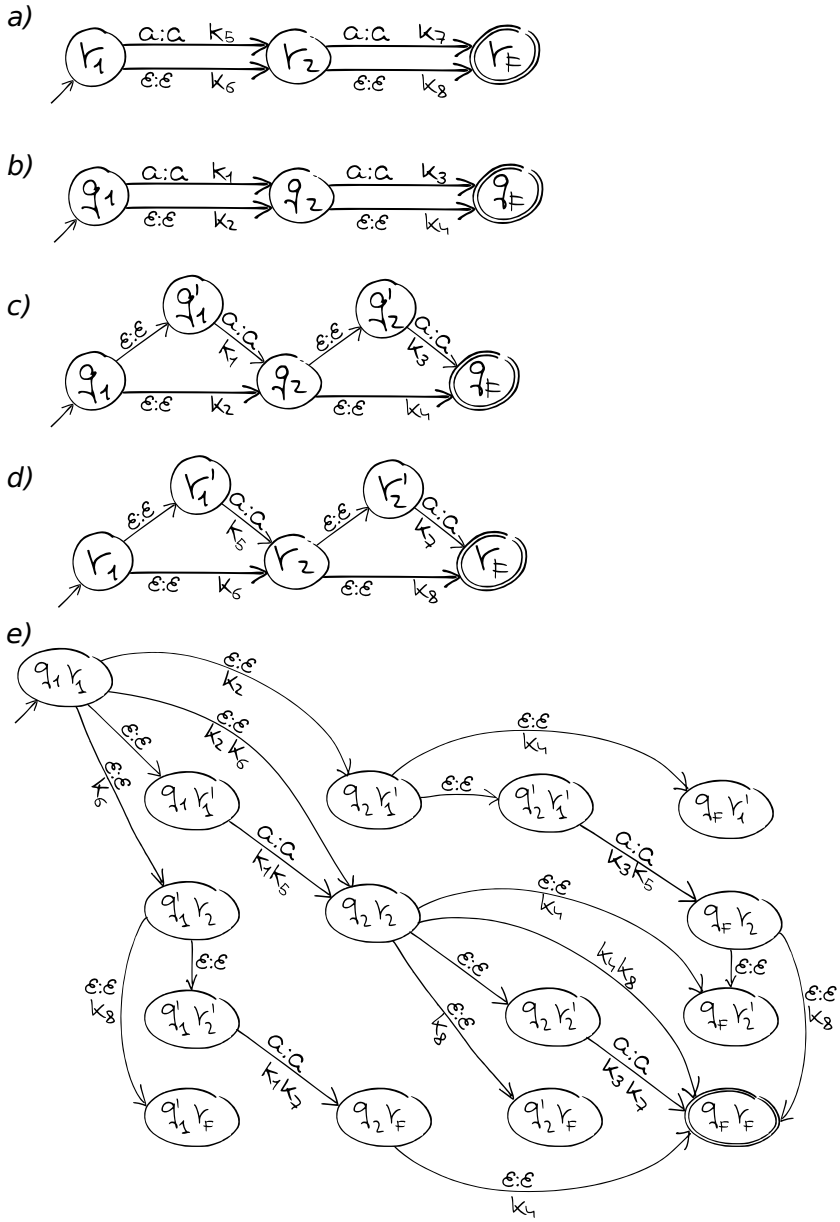


Figure 72: Another example of composition of two FSA with epsilon transitions: *a)* and *b)* are two weighted FSA, *c)* and *d)* are the corresponding models converted into homogeneous epsilon form and, finally, *e)* is the result of composing the models from *a)* and *c)* using Algorithm 2. We can observe that the weighted transducer associated to *a)* (equivalently, to *c*) is $\{\epsilon:\epsilon \cdot k_2k_4, a:a \cdot (k_2k_3 + k_1k_4), aa:aa \cdot k_1k_3\}$, the weighted transducer associated to *b)* (or *d)*) is $\{\epsilon:\epsilon \cdot k_6k_8, a:a \cdot (k_5k_8 + k_6k_7), aa:aa \cdot k_5k_7\}$. The reader can observe that the resulting composition corresponds to weighted transducer *e)* is $\{\epsilon:\epsilon \cdot k_2k_4k_6k_8, a:a \cdot (k_2k_3 + k_1k_4)(k_5k_8 + k_6k_7), aa:aa \cdot k_1k_3k_5k_8\}$.

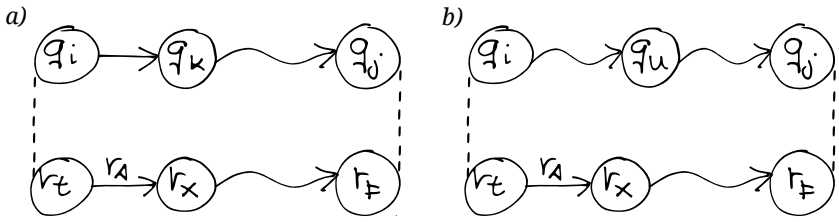


Figure 73: Example of composition of FSA with RTN.

5.5.3 Extension to model reference transitions

The existence of model reference transitions in G_M is what makes it possible to represent not only regular but more general types of CFGs as described in Section 5.4.3.

One of the main advantages of approach parsing of context free models by means of RTNs is that, since FSA can be seen as a restricted case of RTNs where model reference transitions are not allowed, the parsing with finite state models is a particular case of using more general CFGs. Two different and classical approaches can be found in the literature in order to extend the transducer composition algorithms from finite state transducers to RNTs, namely: expansion and memorization. It seems that these extensions have been reinvented several times in the literature. Let us adapt both of them to our constructive exposition.

Expansion

Let us suppose that we are trying to compute $[q_i, q_j] \circ [r_t, r_F]$ and there is a transition from r_t to r_x labeled with $[r_A, r_F]$ (which, in an abuse of notation, we can also depict simply as r_A) as illustrated in Figure 73 *a)*. We would need to simplify, as a sub-problem, the following expression:

$$[q_i, q_j] \circ ([r_A, r_F] \otimes [r_x, r_F]) \tag{5.44}$$

We can expand, using Equation 5.19, the left hand side of the composition provided that $[q_i, q_j]$ is not $[q_F, q_F]$ (in that case, it is equal to $\{\epsilon:\epsilon\} \cdot 1$). Relating the right hand side, instead of expanding $[r_x, r_F]$, as was done in previous cases, let us now expand the transition $[r_A, r_F]$:

$$\left(\bigoplus_{q_k \in Q} tr(q_i, q_k) \otimes [q_k, q_j] \right) \circ \left(\left(\bigoplus_{r_p \in R} tr(r_A, r_p) \otimes [r_p, r_F] \right) \otimes [r_x, r_F] \right) \tag{5.45}$$

As can be observed, transitions of G_I can be consumed without performing any transition in $[r_x, r_F]$, as was the case when combining regular transducers. An special case also occurs when r_z becomes r_F since, in this case, $[r_F, r_F]$ is $\{\epsilon:\epsilon\} \cdot 1$, the identity of the weighted transducer semiring.

Observing Equation 5.45, we can choose among several alternative approaches. For example, we can apply the associative property of $\otimes$ and the distributive property of semiring operators to obtain:

$$\left(\bigoplus_{r_p \in R} \mathrm{tr}(r_A, r_p) \otimes [r_p, r_F]\right) \otimes [r_x, r_F] = \bigoplus_{r_p \in R} \mathrm{tr}(r_A, r_p) \otimes ([r_p, r_F] \otimes [r_x, r_F]) \quad (5.46)$$

where $[r_p, r_F] \otimes [r_x, r_F]$ can be considered as the sum of *paths concatenations* taking one path from r_p to r_F and another from r_x to r_F. By considering this as a conventional span between two states, we could just apply the conventional transducer composition approach described before the introduction of model reference transitions. At a first glance, and since the last state of the first span is always r_F, we could consider, in a new abuse of notation, $\langle r_p, r_x \rangle$ as an artificial state so that we could treat the path concatenation as an ordinary path: The state pair $\langle q_i, \langle r_p, r_x \rangle \rangle$ behaves like the ordinary state $\langle q_i, r_p \rangle$: Its transitions are also obtained by combining transitions from q_i and from r_p. The difference is that it remembers the fact that the path associated to the right hand side has to pursue towards r_x whenever r_F is first reached.

How can this be achieved? It suffices to replace any destination state of the form $\langle q_k, \langle r_F, r_x \rangle \rangle$ by $\langle q_k, r_x \rangle$. As will be clear shortly, this operation consists in popping r_F from the sequence $\langle r_F, r_x \rangle$ to obtain $\langle r_x \rangle \equiv r_x$ as if it was a stack depicted from left to right as from top to bottom. The problem of this approach are nested model reference transitions. They forces us to apply this trick again and again so that the right hand side will model a sequence of path concatenations:

$$[r_k, r_F] \otimes [r_{k-1}, r_F] \otimes \cdots \otimes [r_1, r_F]$$

The information associated to this sequence can be summarized into the tuple $\langle r_k, r_{k-1}, \ldots, r_1 \rangle$. The crucial insight is that it forms a stack data structure with r_k at the top.[58]

The approach described so far is equivalent to the expansion of model reference transitions by a new or fresh[59] copy of the model which is referenced. New states are pushed when traversing model reference transitions and are popped when arriving to r_F, as illustrated in Algorithm 3 and Figure 74.

non-terminating approach

A more serious drawback of this technique is that it can be non-terminating in the general case. Indeed, this approach is aimed at describing the resulting model as a finite state transducer whereas theoretical results [Bar-Hillel *et al.* 1961] have proven that the composition is context free in the general case. Despite this mismatch, the process just described makes sense in at least two particular (non exclusive and, hence, compatible) cases:

58 Indeed, the implementation of this technique proposed later in Chapter 12 will make use of what is known as an spaghetti stack.

59 Using relabeled states.

Data: $G_I : \Sigma^\star \times \Omega^\star \to \mathbb{K}$ a FSA without ϵ transitions,
$G_M : \Omega^\star \times \Phi^\star \to \mathbb{K}$ a RTN without ϵ transitions
Result: $G_O = G_I \circ G_M : \Sigma^\star \times \Phi^\star \to \mathbb{K}$ a RTN without ϵ transitions
$Q' \leftarrow \{\langle q_S, \langle r_S \rangle\rangle, \langle q_F, \langle r_F \rangle\rangle\}$
$tr' \leftarrow \emptyset$
$S \leftarrow$ queue($\{\langle q_S, \langle r_S \rangle\rangle\}$)
def *add_transition(origin* $\xrightarrow{label}$ *dest)*
  if *dest* $\notin Q'$ **then**
    $Q' \leftarrow Q' \cup \{$ dest $\}$
    enqueue(S,dest)
  $tr' \leftarrow tr' \cup \{\text{origin} \xrightarrow{\text{label}} \text{dest}\}$
while not *empty(S)* **do**
  origin $\leftarrow \langle q, \langle r_k, r_{k-1}, \ldots, r_1 \rangle\rangle \leftarrow$ extract(S)
  foreach $r_k \xrightarrow{r_A \cdot k} r'$ **do** `/* assuming that` r_A `is never` r_F `*/`
    dest $\leftarrow \langle q, \langle r_A, r', r_{k-1}, \ldots, r_1 \rangle\rangle$
    add_transition(origin $\xrightarrow{\{\epsilon : \epsilon \cdot k\}}$ dest)
  foreach $r_k \xrightarrow{\{c : d \cdot k_2\}} r'$ **do**
    foreach $q \xrightarrow{\{a : b \cdot k_1\}} q'$ **do**
      if *b=c* **then**
        if $r' = r_F$ **and** $k > 1$ **then**
          dest $\leftarrow \langle q', \langle r_{k-1}, \ldots, r_1 \rangle\rangle$
        else
          dest $\leftarrow \langle q', \langle r', r_{k-1}, \ldots, r_1 \rangle\rangle$
        add_transition(origin $\xrightarrow{\{a : d \cdot (k_1 \otimes k_2)\}}$ dest)
return G_O given by Q' and tr'

Algorithm 3: Composition of a FSA with a RTN usig expansion (this algorithm can be non-terminating for certain inputs).

ACYCLIC INPUTS which implies that the first model is of finite size[60] and, consequently, the composition is also finite[61] and, therefore, regular. Even if this approach makes sense and the complexity is linear with the input length, the size of the second model may grow exponentially with the number of nested expansions, which becomes unpractical except when pruning techniques (to be described in Chapter 10) are used. Nevertheless, the proposed expansion technique is not applicable *as is* in case of (even indirect) left-recursion;

LACK OF RECURSION cycles in the RTN, which means that the model could be flattened into a FSA, proving that it is regular. This means that the resulting model is the composition of two regular models which is also regular.

60 We are ignoring the case of models with loops entirely composed by transitions with no symbol in the common tape.

61 We are assuming that there are no cycles composed uniquely of epsilon transitions in the common tape.

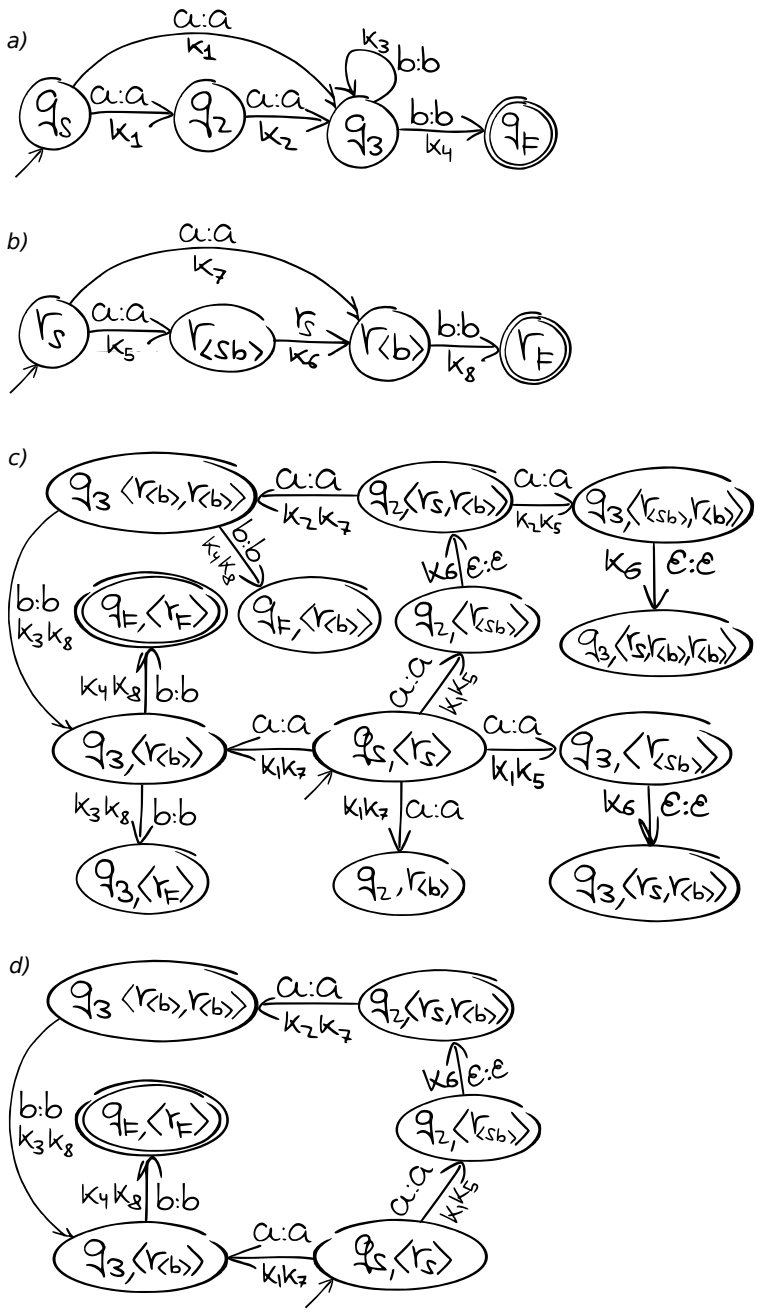


Figure 74: Example of composition of a FSA with a RTN using the expansion approach: a) $G_I = \{((a{:}a \cdot k_1) + (aa{:}aa \cdot k_1^2))\}^{\star}\{b{:}b \cdot k_3\}^{\star}\{b{:}b \cdot k_4\}$, b) $G_M = \{a^n{:}b^n \cdot k_7(k_5k_6)^{n-1}k_8^n | n > 0\}$, c) $G_O = G_I \circ G_M$ obtained with Algorithm 3, and d) the same model after removing non-useful states, we can observe $G_O = \{((a{:}a \cdot k_1k_7) + (aab{:}aab \cdot k_1k_2k_5k_6k_7))(b{:}b \cdot k_4k_8)\}$.

Memorization

The expression of Equation 5.44 can also be tackled by using the paths matching approach: Going back to the example of Figure 73, paths from q_i to q_j are matched with paths, compatible in the common (or intermediate) tape, formed by concatenating one path from $[r_A, r_F]$ (the label of $tr(r_t, r_x)$) with another one from $[r_x, r_F]$.

Each path from q_i to q_j, compatible with another obtained by the aforementioned concatenation, can be neatly divided at an intermediate state because we have assumed that transitions are only labeled with terminal symbols or with the null symbol ϵ. Note that this would not be necessarily true if we allow transitions labeled with more general elements of the weighted language of the common tape.

By hypothesizing the set of possible intermediate states where this division occurs, we can obtain (see Figure 73 *b)*):

$$[q_i, q_j] \circ ([r_A, r_F] \otimes [r_x, r_F]) = \bigoplus_{q_u \in Q} ([q_i, q_u] \circ [r_A, r_F]) \otimes ([q_u, q_j] \circ [r_x, r_F]) \quad (5.47)$$

which remembers us the problem with Equation 5.26 where the same semiring values of the resulting composition were taken into account several times (as shown in Figure 65). Now, the situation seems different since paths have to be matched. Nevertheless, the use of ϵ labels in the common tape may cause problems to this respect, as described below. Let us try to solve the simpler problem without ϵ labels in the common tape to deal later with the general case.

The basic idea consists in describing the system of equations using the appropriate sub-expressions so that this system has the appropriate structure to be described as a CFG in RTN-form. This would provide a constructive technique to obtain the desired composition in a way similar to the state-pair approach of Section 5.5.1. In that section, $\langle q, r \rangle$ was used to denote $[q, q_F] \circ [r, r_F]$ since the second state of each span was always q_F and r_F, respectively. Let us now write $\langle q, q', r \rangle$ to denote $[q, q'] \circ [r, r_F]$ since the span associated to the input FSA G_I does not necessarily finish at q_F whereas the span associated to the G_M RTN model still finishes at r_F.

Using the convention that a set of rules may represent the terms of a summation in a system of equations, we can write the equations derived from $G_I \circ G_M$ by starting from $\langle q_S, q_F, r_S \rangle$ and generating a set of rules using, for each one, one of the following forms:

a) $\langle q_i, q_j, r_t \rangle = \{a{:}b \cdot (k_1 \otimes k_2)\} \otimes \langle q_k, q_j, r_y \rangle$, where we are assuming, in this case, that there is a transition from q_i to q_k labeled with $\{a{:}c \cdot k_1\}$ and another one from r_t to r_y labeled with $\{c{:}b \cdot k_2\}$;

b) $\langle q_i, q_j, r_t \rangle = \langle q_i, q_u, r_A \rangle \otimes \langle q_u, q_j, r_x \rangle$ for transitions outgoing r_t labeled with a model reference, as the transition from r_t to r_x labeled with r_A, as depicted in Figure 73 *b)*. These rules are created for each state q_u reachable from q_i and that can also reach q_j.

Data: $G_I : \Sigma^\star \times \Omega^\star \to \mathbb{K}$ a FSA without ϵ transitions,
$G_M : \Omega^\star \times \Phi^\star \to \mathbb{K}$ a RTN without ϵ transitions
Result: $G_O = G_I \circ G_M : \Sigma^\star \times \Phi^\star \to \mathbb{K}$ a RTN without ϵ transitions
$Q' \leftarrow \{\langle q_S, q_F, r_S\rangle, \langle q_F, q_F, r_F\rangle\}$; $tr' \leftarrow \emptyset$; $S \leftarrow$ queue$(\{\langle q_S, q_F, r_S\rangle\})$
def *add_state*$(\langle q, q', r\rangle)$
  if $\langle q, q', r\rangle \notin Q'$ **then**
    $Q' \leftarrow Q' \cup \{\langle q, q', r\rangle\}$
    enqueue$(S, \langle q, q', r\rangle)$
    if $r = r_F$ **and** $q = q'$ **and** $q \neq q_F$ **then**
      $tr' \leftarrow tr' \cup \{\langle q, q, r\rangle \xrightarrow{\{\epsilon:\epsilon\}} \langle q_F, q_F, r_F\rangle\}$
def *add_transition*$(origin \xrightarrow{label} dest)$
  add_state(dest)
  $tr' \leftarrow tr' \cup \{\text{origin} \xrightarrow{\text{label}} \text{dest}\}$
while not *empty*(S) **do**
  $\langle q, q', r\rangle \leftarrow$ extract(S)
  foreach $q \xrightarrow{\{a:b\cdot k_1\}} q''$ **do**
    foreach $r \xrightarrow{\{c:d\cdot k_2\}} r'$ **do**
      if $b{=}c$ **then**
        add_transition$(\langle q, q', r\rangle \xrightarrow{\{a:d\cdot(k_1\otimes k_2)\}} \langle q'', q', r'\rangle)$
  foreach $r \xrightarrow{r_A\cdot k} r_x$ **do**
    foreach q'' *reachable from* q *and that can also reach* q' **do**
      add_state$(\langle q, q'', r_A\rangle)$
      add_transition$(\langle q, q', r\rangle \xrightarrow{\langle q,q'',r_A\rangle\cdot k} \langle q'', q', r_x\rangle)$
return G_O given by Q' and tr'

Algorithm 4: Composition of a FSA with a RTN, using memorization, when there are no null transitions.

Rules of case *a)* are like the ones also used to describe FSA and correspond to CFG rules of the type $A \to aB$ whereas case *b)* correspond to CFG rules of the form $A \to BC$. This last type makes it possible to interpret the system as a CFG in RTN normal form where $\langle q_F, q_F, r_F\rangle$ is considered as the final state commonly used to convert a right-linear CFG into a FSA. The algorithm to construct the composed model, an extension of the previous one for the case of transducer composition based on the on demand creation of states, is illustrated in Algorithm 4. An example of this composition is shown in Figure 75. The models that have been composed are $G_I = \{(ab)^n \mid n > 0\}$ (illustrated in Figure 75 *a)*) and $G_M = \{a^n b^n \mid n > 0\}$ (the RTN of Figure 63 reproduced again in Figure 63 *b)* for the sake of convenience). Although these models are not transducers *as is*, we are making the abuse of notation described in Equation 5.1 where a language can be seen as a transducer. The model resulting from the composition,[62] associated to the language $\{ab\}$, is shown in Figure 75 *c)*. We can remark the existence of non-useful states that could be removed (Figure 75 *d)*).

62 For transducers obtained from languages (Eq. 5.1), composition becomes intersection.

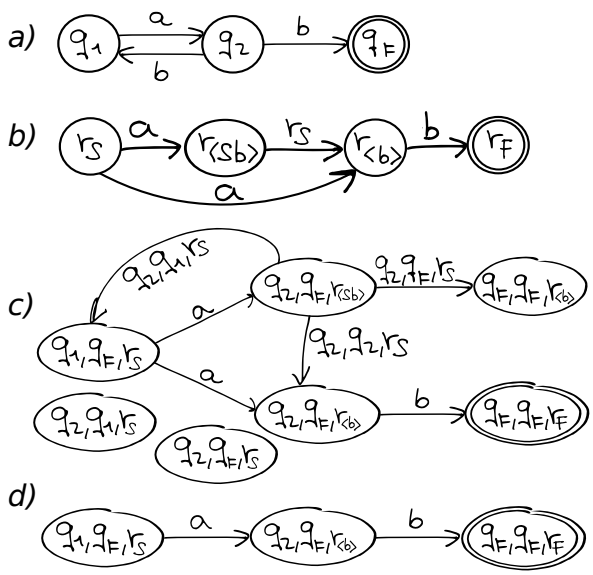


Figure 75: RTN composition example. From top to bottom: *a)* the input model associated to the language $\{(ab)^n \mid n > 0\}$, *b)* the RTN of Figure 63 corresponding to $\{a^n b^n \mid n > 0\}$, *c)* the result of composing $G_I \circ G_M$ using the rules of Equation 5.47 for the case of model reference transitions. Non-reachable states are not depicted. Finally, in *d)* we can observe the result of the composition after removing non-useful states. Transitions are depicted without weights and using the convention that labels of the form "a" from generators, when used as transducers, are interpreted as $a:a$.

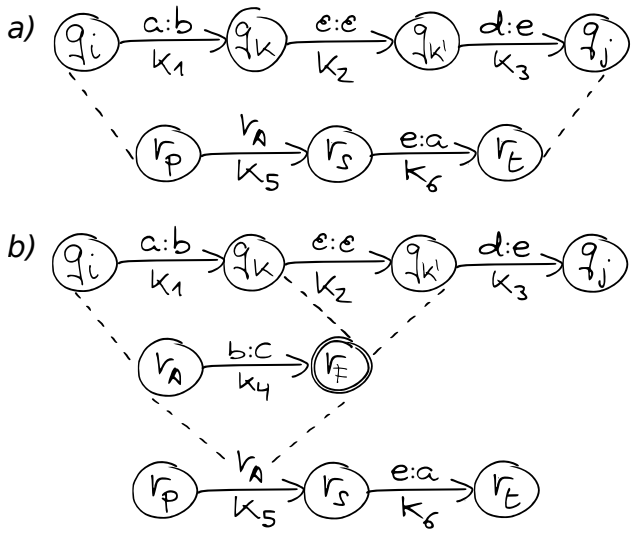


Figure 76: Problems with the presence of reference transitiosn and null symbols. We want to compute the composition of $\{(ad:be) \cdot k_1 k_2 k_3\}$ with $\{(be:ca) \cdot k_4 k_5 k_6\}$. The result is $\{(ad:ca) \cdot k_1 k_2 k_3 k_4 k_5 k_6\}$. The need to compute $[q_i, q_j] \circ [r_p, r_y])$ from Equation 5.44 is illustrated in a). In b) we can see that $[r_A, r_F]$ can be matched with $[q_i, q_k]$ to produce $(a:c) \cdot k_1 k_4$ or with $[q_i, q_{k'}]$ to produce $(a:c) \cdot k_1 k_4 k_2$. In both cases, the final result of $[q_i, q_j] \circ [r_p, r_y])$ is $(ad:ca) \cdot k_1 k_2 k_3 k_4 k_5 k_6$, but allowing both choices would lead to take into account the same value twice, which is incorrect (excepting when using idempotent semirings).

Extension of Memorization to deal with null-transitions

As mentioned before, transitions with the null symbol in the common tape may cause problems if we apply the state-pair approach in a dummy way. A direct application of the summation of Equation 5.47 might take into account the same expression several times in the set of addends, as illustrated in Figure 76, where two different addends (using q_k and $q_{k'}$, respectively) account for the same intermediate string "be". The problem of this procedure, as explained before, is that it ignores the fact that we are departing from an initial expression $G_I \circ G_M = [q_S, q_F] \circ [r_S, r_F]$ and we are just applying transformations.

The problem is aggravated by the fact that reference transitions may represent weighted transducers which contain pairs with the null sequence in the common tape. Put in other words, there may be nullable (w.r.t. the intermediate tape) auxiliary symbols in the CFG associated to the RTN G_M.

The solution adopted here to tackle this problem is essentially the same as when composing two FSA which may have transitions emitting the null symbol in the common tape. This approach, introduced in Section 5.5.4, was based on avoiding the existence of states containing both null and non-null outgoing transitions. The concept of "homogeneous epsilon form" has to be extended from FSA to RTNs and, in order to be practical, we have also to provide some procedure to construct a RTN of this type from an arbitrary RTN. We will delay this procedure to Section 5.5.4 in order to concentrate on the composition algorithm. So, let us characterize FSA and RTNs in "homogeneous epsilon form" by describing the properties they have to fulfill without worrying form the moment in how to achieve them. We can concentrate on proposing an algorithm for composing a FSA with a RTN assuming they have this normal form.

It is first convenient to classify RTN references in a the same way usual transitions are classified into null and non-null depending on the presence of an empty string (denoted by ϵ) in the common tape. A RTN reference is an RTN state that appear labeling some transition or, equivalently, states B of the grammar that appear in rules of the form $A \to BC$). A RTN reference can be classified, according to the weighted transducer it generates, into:

a) emits nothing at all or, equivalently, the weighted transducer they generate is the empty set;

b) only emits pairs with non-empty strings in the common tape, we will call them **non-null RTN references**;

c) only emits pairs with empty strings in the common tape; they will be denominated **null RTN references**;

d) does not fulfill any of the above cases, meaning that something is emitted with and without the empty string in this first tape.

Observe that RTN references of case a) are non-useful and could be removed. Let us remark that even r_F is associated to the weighted transducer $\{\epsilon : \epsilon \cdot 1\}$, which is different from the empty set $\emptyset$, making it to belong to case b). The presence of case d) means that the distinction between null and non-null RTN references is not exclusive.

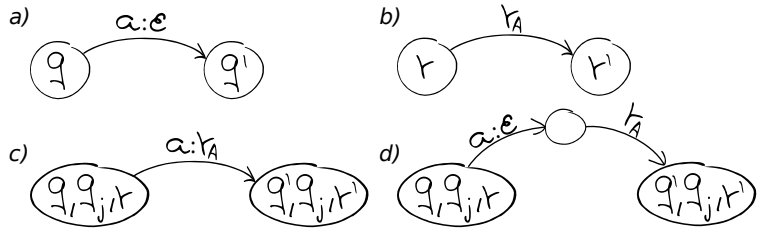


Figure 77: How to combine a null reference with a null reference transition: we would like to combine the null RTN transition of *b)* with the null transition of *a)* as if it were a conventional null transition *c)* (where "$a{:}R_A$" is just an informal way to illustrate what we are trying to mean). However, this is not possible *as is*. However, it can be achieved by using a new auxiliary state *d)*.

definition of homogeneous epsilon form

A RTN is in homogeneous epsilon form when every state r with some outgoing transition[63] falls into the one the following cases:

SYMBOL-EMITTING STATES when all direct transitions outgoing r contain the empty string in the common tape and all reference transitions outgoing r are null RTN references; or

NON-DIRECT EMITTING STATES contains at least one outgoing transition, direct outgoing transitions emit something in the common tape and all outgoing reference transitions are non-null RTN references.

The approach to extend the composition of FSA with RTNs to null transitions is to apply the ideas of Algorithm 2 to adapt Algorithm 4. To this respect, we have to make some observations: 1) firstly, the use of RTNs in homogeneous normal form means that we can assume that null-references are like constant weighted transducers labeling null symbols, while non-null references always consume *at least* one symbol from the common tape in the input FSA; 2) however, null transitions of the FSA cannot be directly combined with null RTN references. Although this trick may be not required, it is easy to see that this issue can be solved with an auxiliary state, as depicted in Figure 77; and, finally, 3) the problems described in Figure 76 have to be avoided. This happens when null transitions in the input model can appear before of after entering or exiting a reference model. We propose to solve it by advancing as much as possible,[64] using null transitions, before entering and before exiting a non-null RTN transition. This means that:

- we cannot leave a RTN model when the FSA destination state has outgoing null transitions.[65] Put in other terms, the state q_j in $\langle q_i, q_j, r_k \rangle$ should never have outgoing null transitions;
- we cannot use a non-null RTN transition from an state q_i of G_I which has null outgoing transitions.

The resulting composition algorithm is shown in Algorithm 5 and an example is shown in Figure 81.

63 Otherwise, an state without any outgoing transition would simultaneously belong to both cases since they become vacuous truth statements. In this way, r_F would not belong to any of these classes and, indeed, it receive a special treatment.

64 Let us remark that we are assuming the lack of cycles comprising uniquely null transitions.

65 This means that, in the example of Figure 76, we would leave k_A at $q_{k'}$, but not at q_k.

Data: $G_I : \Sigma^\star \times \Omega^\star \to \mathbb{K}$ a FSA in homogeneous epsilon form,
$G_M : \Omega^\star \times \Phi^\star \to \mathbb{K}$ a FSA in homogeneous epsilon form
Result: $G_O = G_I \circ G_M : \Sigma^\star \times \Phi^\star \to \mathbb{K}$ a FSA
$Q' \leftarrow \{\langle q_S, q_F, r_S\rangle, \langle q_F, q_F, r_F\rangle\}$
$tr' \leftarrow \emptyset$
$S \leftarrow$ queue($\{\langle q_S, q_F, r_S\rangle\}$)
def *add_state*(st)
  if st $\notin Q'$ **then**
    $Q' \leftarrow Q' \cup \{st\}$
    enqueue(S, st)
    if st *can be expressed as* $\langle q, q, r_F\rangle$ *with* $q \neq q_F$ **then**
      $tr' \leftarrow tr' \cup \{\langle q, q, r_F\rangle \xrightarrow{\{\epsilon:\epsilon\}} \langle q_F, q_F, r_F\rangle\}$
def *add_transition*(*origin* $\xrightarrow{label}$ *dest*)
  add_state(dest)
  $tr' \leftarrow tr' \cup \{\text{origin} \xrightarrow{\text{label}} \text{dest}\}$
def *add_null_RTN_transition*($r_{A\epsilon}$)
  add $r_{A\epsilon}$ and its successors in Q' (if required)
  without including them in S
def *both_non_null*(q, q', r)
  foreach $q \xrightarrow{\{a:b\cdot k_1\}} q''$ **do**
    foreach $r \xrightarrow{\{c:d\cdot k_2\}} r'$ **do**
      if *b=c* **then**
        add_transition($\langle q, q', r\rangle \xrightarrow{\{a:d\cdot(k_1\otimes k_2)\}} \langle q'', q', r'\rangle\}$)
  foreach $r \xrightarrow{r_{A\not\epsilon}} r_x$ **do** `/* r_{A/ε} is a non-null RTN transition */`
    foreach q'' *reachable from* q *and that can also reach* q' **do**
      `/* q'' must be reachable from q after consuming at least one symbol (in the common tape) */`
      if q'' *does not have null outgoing transitions* **then**
        add_state($\langle q, q'', r_{A\not\epsilon}\rangle$)
        add_transition($\langle q, q', r\rangle \xrightarrow{\langle q, q'', r_{A\not\epsilon}\rangle} \langle q'', q', r_x\rangle$)
def *only_r_non_null*(q, q', r)
  foreach $q \xrightarrow{\{x:\epsilon\cdot k\}} q''$ **do**
    add_transition($\langle q, q', r\rangle \xrightarrow{\{x:\epsilon\cdot k\}} \langle q'', q', r\rangle$)

Algorithm 5: Composition of a FSA with a RTN when there may be null transitions (continues on next page).

def *both_null*(q, q', r)
  foreach $q \xrightarrow{\{x:\epsilon\cdot k_1\}} q''$ **do**
    foreach $r \xrightarrow{\{\epsilon:w\cdot k_2\}} r'$ **do**
      add_transition$(\langle q, q', r\rangle \xrightarrow{\{x:w\cdot k_1\otimes k_2\}} \langle q'', q', r'\rangle)$
    if $\exists r \xrightarrow{r_{A\epsilon}\cdot k_2} r'$ **then** `/* ` $r_{A\epsilon}$ `is a null RTN transition */`
      $n \leftarrow$ new_state()
      add_state(n)
      add_transition$(\langle q, q', r\rangle \xrightarrow{\{x:\epsilon\cdot k_1\}} n)$
      foreach $r \xrightarrow{r_{A\epsilon}\cdot k_2} r'$ **do**
        add_null_RTN_transition$(r_{A\epsilon})$
        add_transition$(n \xrightarrow{r_{A\epsilon}\cdot k_2} \langle q'', q', r'\rangle)$
def *only_q_non_null*(q, q', r)
  foreach $r \xrightarrow{\{\epsilon:w\cdot k\}} r'$ **do**
    add_transition$(\langle q, q', r\rangle \xrightarrow{\{\epsilon:w\cdot k\}} \langle q, q', r'\rangle)$
  foreach $r \xrightarrow{r_{A\epsilon}\cdot k} r'$ **do** `/* ` $r_{A\epsilon}$ `is a null rtn transition */`
    add_null_RTN_transition$(r_{A\epsilon})$
    add_transition$(\langle q, q', r\rangle \xrightarrow{r_{A\epsilon}\cdot k} \langle q, q', r'\rangle)$
    `/* ` $r_{A\epsilon}$ `is an improvement over using` $\langle q, q, r_{A\epsilon}\rangle$ `*/`
while not *empty(S)* **do**
  $\langle q, q', r\rangle \leftarrow$ extract(S)
  $non\_null_q \leftarrow \exists$ transition outgoing q with no ϵ in 2nd tape
  $non\_null_r \leftarrow \exists$ transition outgoing r with no ϵ in 1st tape **or**
    $\exists$ a non-null rtn reference outgoing r
  if non_null_q **then**
    if non_null_r **then**
      both_non_null(q, q', r)
    else
      only_q_non_null(q, q', r)
  else
    if non_null_r **then**
      only_r_non_null(q, q', r)
    else
      both_null(q, q', r)
return G_O given by Q' and tr'

(comes from previous page)

5.5.4 Transformation to homogeneous epsilon form

The homogeneous epsilon form has been first described for FSA and has been later extended to RTNs. We have characterized the models with this property (e.g. defining the homogeneous epsilon form) in previous section, but we have not provided a constructive way to obtain such a model from an ordinary FSA or RTN. Moreover, the characterization for the case of RTNs is based on the definition of null and non-null RTN reference transitions and they have been defined in terms of the weighted transducers they generate. We have not yet provided a way to check if a given RTN reference transition belongs to any of these classes.

Transformation of input FSA

Let us first deal with the simpler case of FSA. Without loss of generality, we will only consider the input FSA (the first element in the composition). Relating the second element of this composition, we prefer to describe the case of RTNs which include a FSA model as a particular case. Nevertheless, the first case can be used to transform a FSA used as model just by replacing the tape positions.

The definition of the homogeneous epsilon form for FSA is simpler than the RTN counterpart: there cannot be states which simultaneously have outgoing transitions with and without the null symbol in the common tape.

Fortunately, a simple transformation locally applied on the states not fulfilling this restriction suffices to convert any FSA into the desired normal form. This transformation, depicted in Figure 78, consists in creating an auxiliary state associated to the transitions emitting symbols in the common tape.

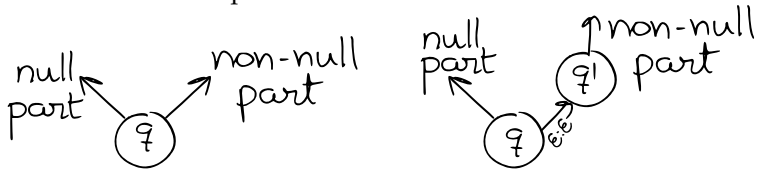


Figure 78: Scheme of the simple homogeneous epsilon transformation.

This simple transformation is applied in Algorithm 6. An example[66] of this transformation, for the models of Figure 67, is depicted in Figure 68. Another example is shown in Figure 71, where case *c)* is obtained from case *b)*.

Transformation of RTN model

The homogeneous epsilon form is slightly more complex for RTNs. In particular, there may be reference transitions representing weighted transducers which contain pairs of strings with and without the empty string in the common tape which, without loss of generality, we will assume that is the first tape.[67]

66 Observe that, when this algorithm is applied to a FSA used in the right side of the composition, the common tape is the first one.

67 RTNs are used as G_M in the computation of $G_O = G_I \circ G_M$.

Data: $G_I : \Sigma^\star \times \Omega^\star \to \mathbb{K}$ a FSA with ϵ transitions
Result: G'_I is G_I in homogeneous epsilon form
$Q' \leftarrow Q$
$tr' \leftarrow \emptyset$
foreach $q \in Q$ **do**
  if $\exists$ *transitions outgoing* q *with and without* ϵ *in 2nd tape* **then**
    $q' \leftarrow$ new_state()
    $Q' \leftarrow Q' \cup \{q'\}$
    $tr' \leftarrow tr' \cup \{q \xrightarrow{\{\epsilon:\epsilon\}} q'\}$
    foreach $q \xrightarrow{\{x:y\cdot k\}} q''$ **do**
      if $y = \epsilon$ **then**
        $tr' \leftarrow tr' \cup \{q \xrightarrow{\{x:\epsilon\cdot k\}} q''\}$
      else
        $tr' \leftarrow tr' \cup \{q' \xrightarrow{\{x:y\cdot k\}} q''\}$
  else
    copy transitions outgoing q from tr into tr'
return G'_I given by Q' and tr'

Algorithm 6: FSA transformation into homogeneous epsilon form.

gen. null	gen. non-null	type of state
		non-useful
✓		non-direct emitting
	✓	symbol-emitting
✓	✓	mixed

Table 4: Classification of RTN states.

We have seen that RTN references can be classified into:

- those that emit nothing at all are non-useful;
- null RTN references;
- non-null RTN references; and, finally
- those which generate a weighted transducer containing both null and non-null strings in the common tape.

Relating states in general (not only those used to label RTN references), we can assign the following boolean values:

GENERATE NULL means that the transducer generated from this state contains at least one string pair with the empty string in the common tape; and

GENERATE NON-NULL indicates that at least one of the string pairs generated from this state has a non-value in the common tape.

States can be classified in the same way as the particular subset of RTN references according to these boolean values as shown in Table 4. It is possible to perform this classification as a sequence of approximations in an iterative way (not very different from the computation of the set of nullable non-terminals of a CFG) as explained in Algorithm 7.

Data: $G_M : \Omega^\star \times \Phi^\star \to \mathbb{K}$ a RTN possibly with ϵ transitions
Result: Classification of G_M states (2 mappings $Q \to$ Boolean)
$g\_null \leftarrow mapping(Q, Bool)$
$g\_non\_null \leftarrow mapping(Q, Bool)$
foreach $r \in Q$ **do**
  $g\_null[r] \leftarrow$ False
  $g\_non\_null[r] \leftarrow$ False
$g\_null[r_F] \leftarrow$ True
$S \leftarrow$ queue($\{r_F\}$)
while not *empty(S)* **do**
  $r' \leftarrow$ extract(S)
  useful $\leftarrow g\_null[r']$ **or** $g\_non\_null[r']$
  foreach $r \xrightarrow{\{\epsilon : b \cdot k\}} r'$ **do**
    if not $g\_null[r]$ **and** $g\_null[r']$ **then**
      $g\_null[r] \leftarrow$ True
      enqueue(r)
  foreach $r \xrightarrow{\{a : b \cdot k\}} r'$ **do**
    if not $g\_non\_null[r]$ **and** *useful* **then**
      $g\_non\_null[r] \leftarrow$ True
      enqueue(q)
  foreach $r \xrightarrow{\{r_A \cdot k\}} r'$ **do**
    if not $g\_null[r]$ **and** $g\_null[r_A]$ **and** $g\_null[r']$ **then**
      $g\_null[r] \leftarrow$ True
      enqueue(r)
    if not $g\_non\_null[r]$ **and** $g\_non\_null[r_A]$ **and** *useful* **then**
      $g\_non\_null[r] \leftarrow$ True
      enqueue(r)
return g_null, g_non_null

Algorithm 7: Classification of RTN states.

Our main aim is to avoid the last type of states when used as RTN references. Given a reference r_A of this last type, we will try to split them into two disjoint sets $r_{A\epsilon}$ and $r_{A\not\epsilon}$ such that $r_A = r_{A\epsilon} + r_{A\not\epsilon}$, being $r_{A\epsilon}$ a null RTN reference and $r_{A\not\epsilon}$ a non-null RTN reference, respectively, as illustrated in Figure 79 *a)*. It is very easy, using Algorithm 7 and Table 4, to detect those mixed emitting reference transitions r_A that have to be divided.[68]

It is also easy to construct $r_{A\epsilon}$ by keeping the null transitions and null reference transitions outgoing r_A towards destination states also containing the generate null flag active, as depicted in Figure 80 *a)* and *d)*. Cases *b)* and *c)* are forbidden. When the destination state also contains the generate non-null active (see case *d*) have to be split it, as depicted in case *h)* and shown in the example of Figure 82.

68 Other mixed states would have to be divided as a consequence of applying rule *g)* of Figure 80, as explained below.

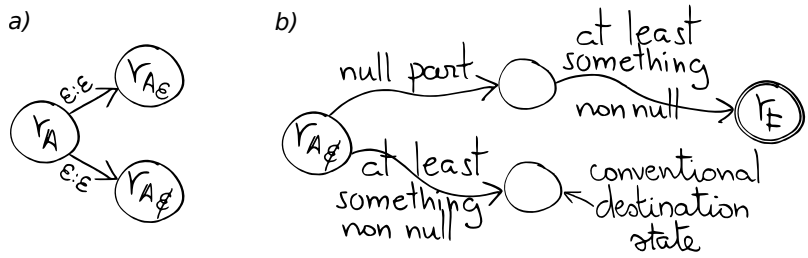


Figure 79: Scheme for splitting a mixed RTN transition: *a)* how a mixed reference transition r_A can be split into $r_{A\epsilon} + r_{A\not\epsilon}$, and *b)* the construction of $r_{A\not\epsilon}$ is more difficult than that of $r_{A\epsilon}$.

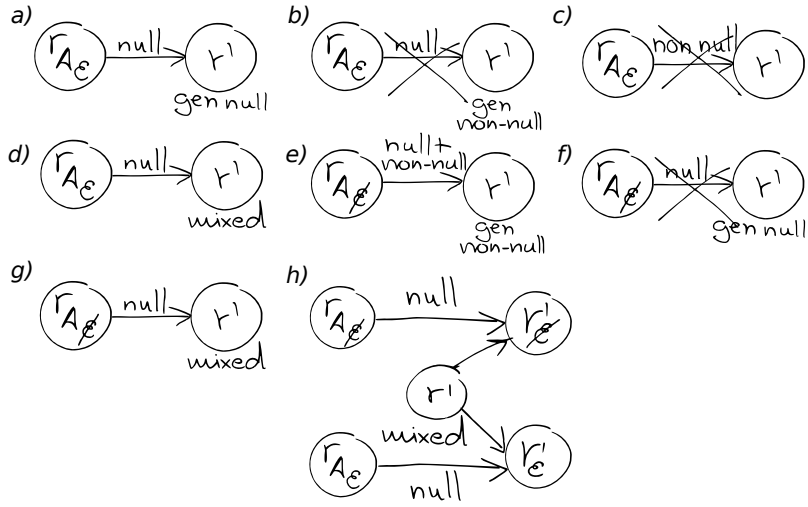


Figure 80: Some allowed and forbidden cases appearing in the RTN homogeneous epsilon transformation depending on the type of arcs and the type of destination states: *a)* the easy case for $r_{A\epsilon}$, *b)* and *c)* are forbidden cases whereas *d)* is not valid and the destination state should be also split, as illustrated in case *h*. Regarding the rules for $r_{A\not\epsilon}$, any transition to a non-null destination state is valid (case *e)*) and case *f)* is the one forbidden. When the transition is null and destination state is mixed (case *g)*), we have also to split the destination state as indicated in case *h)*.

The construction of $r_{A\not\epsilon}$ is considerably more complex since it can contain null parts but, at the same time, we have to guarantee that paths comprising only such null-parts cannot reach r_F. The idea to perform this construction is depicted in Figure 79 *b)*.

It is possible to construct $r_{A\not\epsilon}$ by applying the rules of Figure 80: when the destination state only has active the generate non-null flag, we can place the transition no matter if it is null or not (case *e)*). The only difficult case is when pointing a null transition towards a mixed destination state r' (case *g)*). In that case, we split r' and the transition points towards $r'_{\not\epsilon}$ (case *h)*).

Finally, when no mixed RTN transitions remain, we proceed to apply the scheme of Figure 78 as in Algorithm 6. An example of a RTN before and after the homogeneous epsilon form transformation is depicted in Figure 82.

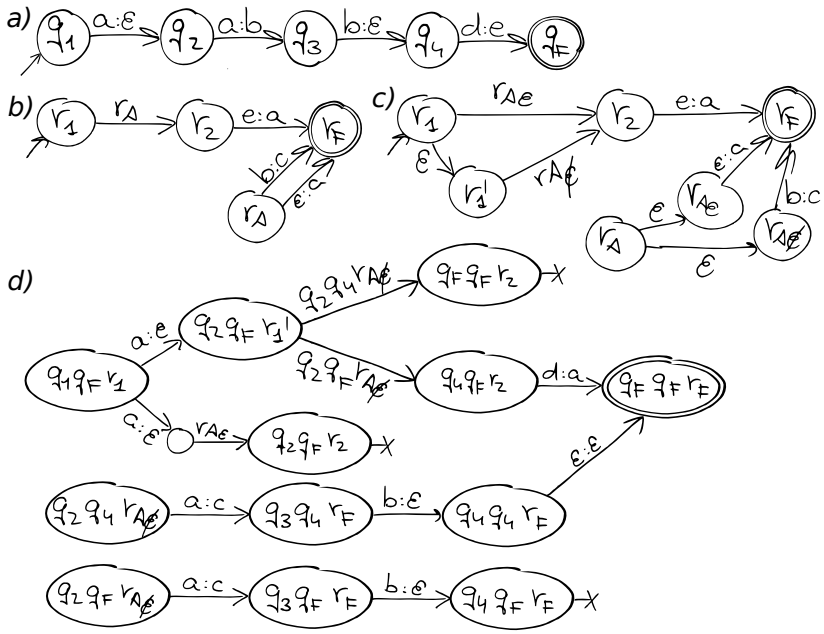


Figure 81: Example of composition of RTN with FSA using Algorithm 5.

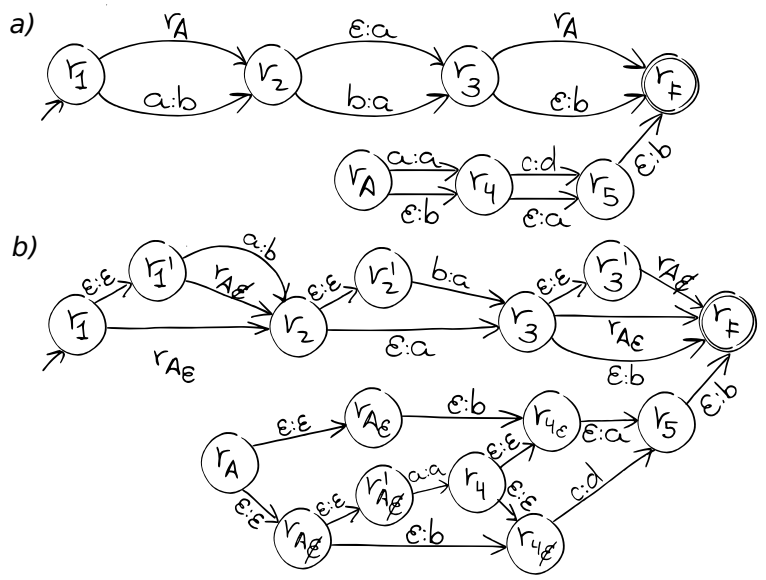


Figure 82: Example of RTN homogeneous epsilon transformation: *a)* a RTN with just one reference transition, R_A, that emits string pairs with and without the null string in the first component of the pair, *b)* the same model after applying the homogeneous epsilon transformation. $r_A = r_{A\epsilon} + r_{A\not\epsilon}$, where $r_A = \{ac{:}adb + a{:}aab + c{:}bdb + \epsilon{:}bab\}$, $r_{A\epsilon} = \{\epsilon{:}bab\}$ and, finally, $r_{A\not\epsilon} = \{ac{:}adb + a{:}aab + c{:}bdb\}$. We can also observe in *b)* that the entire model is converted into homogeneous epsilon form by using $r_{A\epsilon}$ and $r_{A\not\epsilon}$ as ordinary transitions emitting or not the null string in the common tape, splitting a state with the scheme of Figure 78 when required.

5.6 REVIEW OF SOME (CLASSICAL) PARSING APPROACHES, DECODERS AND ALGORITHMS

This section reviews some parsing approaches, limited to CFGs and FSA, which are exemplified by some well known parsing algorithms. The reason to postpone this section after having described our composition algorithms has been to provide a fresh and unbiased proposal without resorting to more classical concepts which, in our humble opinion, would have complicated the previous exposition. Before entering the review, let us summarize the exposition made so far:

- we have basically seen weighted languages as values which can be expressed by means of systems of equations; and
- we have derived a composition by obtaining a system of equations for the composed model.

And this has been done without using concepts like parse trees, derivations, dotted rules and the like.

particular & limited scope

We should also take into account that our goals and scope may be different from most works on parsing: While they are usually limited to consider the generation of strings by an unweighted CFG, our aim is to combine two different weighted languages in order to solve the "classical problems" described in previous chapter (Section 4.3). This implies the following extensions:

- weighted grammars;
- the input is no longer an string but a word graph.

overwhelming possibilities

Let us now proceed with the review mainly intended to contextualize our proposal and for the sake of completeness. Only a sketch is given since the number of parsing algorithms (and relationships, histories, details...) is overwhelming (for more information, the reader is referred to books and surveys such as [Grune and Jacobs 2007; Aho *et al.* 1986]).

parsing as search

Classical parsing algorithms seem focused on recognition, i.e. they seek to obtain a suitable derivation for the input sentence. Nevertheless, it is often possible to obtain parse trees by means of a post-processing of the data structures generated during this process. In an abuse of notation, the term parser will also be used in these cases. Most algorithms are compatible with viewing parsing as a search process. With this setting in mind, there have been many attempts to characterize parsing algorithms along several dimensions, the most notable ones are:

BOTTOM-UP VS TOP-DOWN while top-down parsers starts from the axiom and construct a parse tree trying to reach the symbols of the input sequence, the bottom-up counterparts seem to start from the leaves of the tree processing the symbols of the sentence;

DIRECTIONAL VS NON-DIRECTIONAL refers to the order in which the elements of the input sequence are processed. While directional methods proceed in a left-to-right (alternatively, right-to-left) order, non-directional methods may process the input symbols in a random way requiring the entire sequence from the beginning.

They can also be characterized by the type of grammar[69] they are able to handle. Some algorithms cannot deal with empty productions, unary productions, (even indirect) left-recursion and so on.

Observe that, although the use of semirings to describe parsing seems to subsume the recognition approach,[70] the opposite is not true since there are recognition and parsing algorithms well suited for determining acceptance that cannot be generalized in a direct way to the weighted (e.g. probabilistic) case.

recursive descent

Perhaps, the most paradigmatic example of parsing as a search is the recursive descent parsing, which is a directed top-down method based on backtracking. A different function is usually associated to each non-terminal. These functions have a different behavior for each possible right-hand side: they perform the matching with the symbols of the input sequence and call functions associated to non-terminals of this right-hand side in the appropriate order. A naive top-down parser is not only unable to deal with left-recursion but is also very inefficient due to the huge search space and the repeated computation of the same sub-goals: When it terminates, the cost may be exponential with the input size.

LL parsers

There are very efficient top-down predictive parsers for some CFG restricted subsets: $\mathrm{LL}(k)$ grammars are those that can be parsed without backtracking when taking the next k tokens into account. We will rather focus on parsing techniques for general CFGs.

The exponential cost can be transformed into polynomial by means of memoization [Michie 1968], based on the idea of storing common sub-problems to be reused later, as done in functional programming.

Unger

Another example of a top-down parser, in this case non-directional, is the Unger parser [Unger 1968] which splits the input sequence into as many parts as the rank of the production rule it is trying to apply, leading to several sub-problems of shorter size (a divide-and-conquer approach). This naive technique can be adapted to deal with empty transitions. The exponential cost can be transformed into polynomial by means of memoization. However, this cost is still $O(n^{r+1})$, being n the size of the input sentence, and r the rank of the grammar (a feature shared by other parsing algorithms). A cubic cost can be often obtained in these cases by means of an implicit or explicit binarization of the grammar.

The idea of using a data structure to keep track of common subgoals has been independently proposed by several researchers during the 1970's. A data structure known as Well Formed Substring Table (WFST[71]) or chart can be viewed as a graph representing the input sentence (i.e. edges represent terminal symbols and vertices positions between symbols) and their possible segments (i.e. extended with edges associated to the transitive closure). These data structures and techniques are at the core of dynamic programming (DP). General parsing methods based on DP are generally known as tabular parsers; among

69 In some cases, they assume a pushdown automaton.

70 We can use the boolean semiring to determine acceptance or change to other semiring types for computing the probability of the best derivation, the set of derivations and so on (see Table 3).

71 This term can be traced back at least to [Kuno and Oetinger 1962]. Observe also that the acronym is in clash with "Weighted Finite State Transducer". However, we will use WFST to refer to Well Formed Substring Tables in the rest of this section.

them, we can also mention Cocke–Younger–Kasami (CYK) [Cocke and Schwartz 1970; Younger 1967; Kasami 1965] and Earley [Earley 1970] parsers described below.

CYK non-directional bottom-up

CYK is one of the most popular parsing algorithms based on DP. It is a non-directional bottom-up algorithm that has been discovered independently *at least* three times,[72] and characterized by being restricted to work with CFGs in Chomsky Normal Form[73] (CNF) and by computing the sets $\{A|A \in \mathrm{N}, A \xrightarrow{\star} w\}$, for all sub-words w of the sentence to be parsed, from shorter to longer spans: The WFST, table or chart where spans are stored is often assimilated to a triangular matrix[74] of size the sentence length *plus one*.[75] Table cells associated to spans of length one are first completed using the input sentence and the rules of the form $A \rightarrow a$, whereas rules of the form $A \rightarrow BC$ are applied to fill the cells of length greater than one, from lower to higher length, afterwards.

The chart encodes information which can be used to obtain a packed parse forest, but this structure itself is not a parse tree or a packed shared forest. In this way, CYK is not a parser but a recognizer (unless the second extraction stage is taken as part of the algorithm): The membership of the sentence to the grammar can be easily determined by checking whether the full sentence is associated to the axiom or not.

inside algorithm

This description, only suited for the boolean semiring, can be extended to other semiring types by associating to each possible combination or tuple *(sentence span, non-terminal symbol)* a semiring value. This extension leads to the CYK variant known as inside algorithm.[76]

chart parsing

Chart parsing, proposed in [Kay 1980], is based on the idea of extending the use of WFSTs to store not only finished sub-goals (called finished edges) but also active edges (goals that have been discovered but are not yet finished) that are stored in an agenda used to specify the order in which tasks are applied.[77] Chart parsing has been extended later to the probabilistic case [Klein and Manning 2001a].

Relating bottom-up techniques other than CYK, let us focus for a moment on directional methods. If we try to recognize as reversing the derivation process, we can assume that the input sentence was pro-

72 *At least* three times since we do not know how many students have also reinvented it. The reader is kindly invited to visit:
`http://en.wikipedia.org/wiki/Multiple_discovery` and
`http://en.wikipedia.org/wiki/List_of_multiple_discoveries`.

73 Described in Section 5.4.1. Nevertheless, some less restrictive variants have also been proposed such as [Leermakers 1989] or [Lange and Leiß 2009] where a description for grammars in 2NF is provided (this normal form has been described in Section 5.4.3 related to the proposed description of a RTN as a way of describing a CFG).

74 This chart or table is, indeed, three-dimensional since each span contains a different value for each possible non-terminal. A triangular matrix representation is a very ad hoc way to represent a WFST to the case of input sequences. A bad generalization from sentences to DAGs using the triangular matrix (instead of the concept of transitive closure of the graph), as described in [Chappelier *et al.* 1999], leads to a search space which can be much larger than necessary, as pointed out in [España-Boquera *et al.* 2007].

75 Because chart items are associated to input DAG vertices. The description of parsing where the input is a FSA is, undoubtedly, not only more general but also more elegant.

76 A name related to the inside and outside probabilities.

77 The idea of separating the specification of the search space and how it is explored is often related to the use of inference rules and deduction [Sikkel 1993; Shieber *et al.* 1995] later extended to more general semirings and weighted logic [Goodman 1999]. Indeed, the term "semiring parsing" used nowadays is not associated to the use of semirings for parsing purposes but implicitly assume the use of weighted logic deduction.

duced from the grammar by means of rightmost productions. This assumption allows us to reduce (i.e. to replace the right-hand side of a rule by its non-terminal) the input sentence from left to right. We can divide, at any moment, the sentential form in two parts: a being-reduced part, on the left, and a suffix of the terminals not taken yet into account, on the right. The left part comprises both terminal and non-terminal symbols and is often represented by means of a stack since the reductions are applied to the right-most or top of this stack.

shift-reduce

Besides this reduction operation, it is also possible to shift a terminal symbol from the input stream to the top of stack, hence the name of this methodology: shift-reduce parsers. Most popular shift-reduce parsers are deterministic and create a table of actions, as is the case of LR parsers [Knuth 1965] and some of their variants (LALR, SLR) which are limited to work with some specific CFG families.[78] Nevertheless, it

GLR

is possible to use this technique for general CFGs by means of a Generalized LR parsing (GLR) proposed in [Tomita 1985], although the description of algorithms for parsing non-deterministic push-down automata can be traced back to [Lang 1974]. This parser, with the same worst-case cubic complexity[79] of CYK, makes use of non-deterministic models (i.e. several actions could be performed in some cases) and tries all such actions in parallel (in a breadth-first manner). In order to be able to use the stack in all such parallel search processes, a persistent[80] data structure version of stack, known as graph-structured stack (GSS), is used. A key feature of this parser is that these parallel searches can converge and their associated stacks are recombined.

GLL

If GLR was the extension of LR parsing to general CF languages, it is also possible to generalize LL parsers to the general case, leading to the Generalized LL (GLL) parser [Scott and Johnstone 2010] based on the recursive descent technique but executing parallel searches in case of non-determinism. This parser runs also in a worst-case cubic time and makes use of (a modified version of) the GSS.

Besides bottom-up and top-down parsing strategies, there are mixed

left-corner

approaches as is the case of the left-corner parsing [Rosenkrantz and Lewis 1970] which, as its name suggests, is based on the left-corner relation (a concept introduced in Section 5.4.3) and that has been generalized afterwards [Nederhof 1993].

Earley

Earley recognizer/parser[81] [Earley 1970] is a top-down[82] DP algorithm that constructs a table in a left-to-right traversal of the input sequence. Each table element is associated to an input sequence position and contains a set of items that summarize the information about all the left-most derivations that are compatible with the input up to this position. Each of these items (that we can depict $i : {}_k A \rightarrow \alpha \cdot \beta$ following, for instance, [Stolcke 1995]), in turn, contains:

78 Related to deterministic context-free languages which constitute a proper subset of context-free languages. See [Grune and Jacobs 2007; Aho *et al.* 1986] for more details.

79 As with many other parsers, the cost is $O(n^{r+1})$, being n the size of the input sentence, and r the rank of the grammar, but the grammar could be binarized.

80 The distinction between persistent and ephemeral data structures is better explained elsewhere in this work in Section 10.2.2.

81 As with CYK and other parsing algorithms, the basic description provided here is a recognizer that can be extended or modified to become a parser.

82 Most works describe Earley parser as being purely top-down, others remark that it is top-down with bottom-up recognition.

- a dotted-rule $A \to \alpha \cdot \beta$ (see Section 5.4.3) which we know not only which rule is being used in the derivation but also which portion of the right-hand-side has been parsed (α) and which part remains to be parsed (β);
- the input position k at which the non-terminal A has started;
- observe that the value i representing portion of the input sentence which has been parsed is not explicitly stored since it can be deduced from the table position where the item is stored;
- versions other than the basic recognition may require some semiring values attached to each item.[83] For instance, the probabilistic Earley parser [Stolcke 1995] store forward and inner probabilities.

Initial versions of Earley algorithm may take look-ahead tokens into account but this feature is generally avoided and we have skipped it.

The parser traverses the input sequence and, at each position, traverses the items of the table from the corresponding position applying one of the following operations:

PREDICTOR is applied to items of the form $i : {}_kA \to \alpha \cdot B\beta$ where there is a non-terminal B after the dot. Its purpose is to add new items of the form $i : {}_kB \to \cdot\mu$ for each CFG rule $B \to \mu$ representing potential expansions of the auxiliary or non-terminal symbol B;

SCANNER is applied to items of the form $i : {}_kA \to \alpha \cdot a\beta$ where a is a terminal symbol that coincides with the one on the input at position i. This operation inserts the item $i+1 : {}_kA \to \alpha a \cdot \beta$ at the following position;

COMPLETER takes care of the use of rules which have been completely traversed by the dot (e.g. $i : {}_jB \to \mu\cdot$, with $j \leqslant i$) in order to obtain, for each other item of the form $j : {}_kA \to \alpha \cdot B\beta$, a new item of the form $i : {}_kA \to \alpha B \cdot \beta$.

Observe that items are never removed and are only added either to the set associated to the current position i (at the end) or to the next position $i+1$. This algorithm is initialized by including an item representing the fact that the start symbol or axiom S has started at the beginning of the sequence. At the end, it suffices to check the presence of an item, in the last position of the table, with a rule of S with a dot at the end. Parsers that obtain the set of derivations (e.g. by means of packed shared forests) can be obtained from this recognizer.

Some modifications of the algorithm have been proposed in the literature. One of the most known ones is due to [Graham *et al.* 1980] where null and unary productions are pre-computed. An extension to deal with stochastic CFGs and to compute prefix probabilities is described in [Stolcke 1995]. Earley parser has also been extended to deal with EBNF grammars [Wang and Jin 2006] and grammars with regular right-hand sides [Jim and Mandelbaum 2010]. It can also be adapted to take profit of partially bracketed input sequences [Stolcke 1995; Sect. 5.3].

83 In other case, it is trivially the true value of the Boolean semiring, that is therefore obviated.

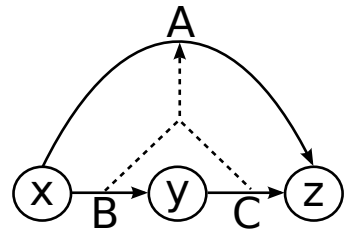


Figure 83: Hypergraph vertices represent spans of the transitive closure of the input sequence (an acyclic Mealy FSA). This hypergraph edge represents the application of the $A \rightarrow BC$ production rule (inspired by [España-Boquera *et al.* 2007; Fig. 4*a)*]).

weighted logic deduction

As proposed in [Sikkel 1993; Shieber *et al.* 1995], inference rules and logic deduction can be used to describe parsers manipulating items. From a set of initial items, the parsing process applies rules to reach a goal. It is possible to describe in this way classical parsing algorithms such as CYK or Earley. This approach has been extended to the weighted case and is usually associated to the use of semiring values to the extent that the term "semiring parsing", as coined by [Goodman 1999], implicitly means using also weighted deduction.

shortest paths

As explained in Section 5.5.1, the use of semirings is related to the concept of shortest paths. It is widely known that the decoding problem on WFSTs corresponds to the computation of the shortest distance in a graph, known as *trellis*, associated to the composition of the input string with the WFST model. This shortest distance is not necessarily related to our intuitive notion of "shortest" unless using a semiring with a total order defined and, as is usually the case, with other properties known as monotonicity and superiority (see Section 5.3). These properties allow optimizations other than the associativity and distributive so that shortest paths can be computed by means of dynamic programming when the input is acyclic (as is the case of strings and even of word graphs), by means of Dijkstra algorithm [Dijkstra 1959] in the presence of cycles[84] or even with $A^\star$ search strategies.

hypergraph exploration

However, when changing from WFSTs to more general[85] CFGs, a graph does not suffice to express the search space and we need to resort to hypergraphs. Hypergraphs constitute a generalization of graphs where edges may connect several vertices. In particular, the search space for parsing with CFGs is based on directed hypergraphs [Gallo *et al.* 1993; Klein and Manning 2001b; Huang 2008a; España-Boquera *et al.* 2007] where each hyperedge has one head but may possibly have several tails, as shown in Figure 83. It is possible to associate arbitrary n-ary functions to hyperedges having n tails. Indeed, when these functions fulfill certain properties,[86] it is possible to adapt DP based Viterbi algorithm, Dijkstra algorithm, and $A^\star$ search techniques to this scenario [Knuth 1977; Nederhof 2003; Huang 2008a].

packed shared forests

The result of parsing, in the form of a compact representation of the set of feasible parse trees, is often represented by means of *packed shared forests* [Billot and Lang 1989] which are essentially acyclic hypergraphs, but are also related to and/or graphs,[87] acyclic CFGs and acyclic RTNs.

84 When the semiring fulfills the superiority property, which is often the case when using positive weights as in the *real* semiring. Remember that cycles might appear, even if the input data is an acyclic model, due to null transitions.

85 Since regular models are a proper subset of context free ones.

86 Namely, to be a superior function, see [Knuth 1977; Nederhof 2003; Huang 2008a].

87 A.k.a. $\otimes/\oplus$-graphs or *mul/add* graphs when using semirings other than the Boolean.

hypergraphs may be overkill

Although the hypergraph framework is very general, their capability to use arbitrary n-ary functions at hyperedges is overkill when only the semiring $\otimes$ operator is used in practice: One the one side, the size of the hypergraph search space grows exponentially with the rank of the grammar (as mentioned in this section when discussing other parsing approaches) so that, in practice, the grammar should be binarized (explicitly beforehand or implicitly on-the-fly). On the other side, viewing the search space as an and/or (or mul/add) graph, allows further factorization possibilities which are not exploited by the general hypergraph framework.

parsing as intersection

Parsing as intersection is an elegant way to generalize parsing from input sequences not only to word graphs but to FSA. Originally described for intersection of languages [Bar-Hillel *et al.* 1961], it has been later extended to deal with probabilities [Nederhof and Satta 2003] and top-down filtering [Nederhof and Satta 2008]. A more recent work also extends the algorithm to include semiring weights and top-down filtering but employs weighted deduction to describe the algorithms [Dyer 2010; Section 2.3]. [Hanneforth 2011] describes an intersection for weighted CFGs by converting them first into an automaton as those used in LR parsing.

sub-cubic cost

[Valiant 1975] showed that the computations of CYK parser can be reduced to Boolean matrix multiplication. This has lead to sub-cubic parsing algorithms. The crux to achieve this cost is the use of sub-cubic matrix multiplication algorithms, as is the case of the Strassen method [Strassen 1969], although even more recent methods have achieved lower bounds on the exponent of the asymptotic cost of square matrix multiplication.[88]

unconventional parsing approaches

There are other parsing algorithms that seem to be outside previous categories. Let us see two different examples: One of them is the iterative use of transducer composition until reaching a fixed point [Roche 1994; Laporte 1996]. [Quint 2004] claims to be able to extend this technique to the weighted case. Another non-conventional cubic parsing approach for (probabilistic) CFGs, proposed in [Hulden 2011], takes profit of the fact (Chomsky–Schützenberger theorem) that any CFG can be expressed as a combination of a regular grammar and a grammar of balanced parenthesis (a bracketed or Dick language, see Section 5.4.1).

CFGs in ASR

Relating the relationship between parsing of regular and CFGs, we can observe that classical parsing algorithms in both fields often uses different concepts and nomenclature. The extension of ASR decoders from regular to CFGs may provide some attempts to close the gap between them: [Dupont 1993; 1996; Kita *et al.* 1989; Brugnara and Federico 1997; Schalkwyk *et al.* 2003; Thomae 2006] are some examples of the extension of the classical Viterbi algorithm to the case of CFGs. It is worth mentioning that all of them adopt the approach of expansion, essentially similar to the recursive descent whose cost is not bounded with a polynomial cost. Relating this fact, it is explained in [Dupont 1993] that the use of pruning techniques (Section 10.6.1) may control the cost in practice. The decoding approach described

88 Coppersmith–Winograd algorithm [Coppersmith and Winograd 1987] was the asymptotically fastest known algorithm until 2010 and has been more recently been superseded [Gall 2014].

in [Dupont 1993; 1996] is based on a transformation of models into Greibach normal form (see Section 5.4.1) and the modification of the underlying model representations on demand. Other works adopt the use of spaghetti stacks instead in order to release the memory once the expanded model is no longer needed. The use of Earley or similar parsers in ASR is not so common: An example is [Jurafsky *et al.* 1995], where the capability of the parser from [Stolcke 1995] to compute prefix probabilities is used to obtain word-transition probabilities.

To summarize, this section has just brought a short overview or a glimpse of most known classical parsing algorithms, bypassing many details[89] and leaving aside lots of interesting concepts, issues and subfields such as, for instance, partial parsing (island parsing,...), the inclusion of error models (edition operations), etc.

5.7 FROM COMPOSITION TO RECOGNITION & DECODING

The purpose of this short section is not to discuss the relationship and the differences between transducer composition and recognition or decoding. This difference has been already explained in Section 5.2. The parsing approach based on *transducer composition*[90] was explained in that section and reviewed again, together with other approaches, in Section 5.6. These ideas are not new, since they can be traced back at least to [Bar-Hillel *et al.* 1961; Lang 1988; 1994]. In this paradigm, the input sentence is first converted into an input automaton that only accepts/emits this particular string. This has several advantages since the input string can be generalized to deal with word graphs or even to construct models generating infinite languages in order to easily compute, for instance, the probability of a prefix, a suffix and so on.

These advantages are important in our setting, which is related to the use of TSGMs: the input is no longer a sequence but a weighted acyclic graph implicitly representing lots of segmentation hypotheses together with the corresponding likelihoods.

Section 5.5 has been devoted to provide a point of view of transducer composition extended from regular to context free models including novel algorithms to deal with the weighted case even in the presence of null transitions, and even when using non-idempotent semirings.

The main purpose of this section is to remark that, often, the desired result is not a composed RTN but the one-best solution (or the set of N-best solutions together with the respective scores). So, instead of applying the proposed composition algorithms *as is* in order to obtain the best solutions, it is better to directly compute what we are looking for. Put in other words, we would like to emphasize that these composition algorithms can be specialized, in some cases, to better

89 To the extent that some parsing algorithms have only been mentioned, without further explanation (e.g. left-corner parsing) or simply omitted. As stated at the introduction of this section, more details can be found in the provided literature and references therein.

90 That may be also better known as *parsing as intersection*. However, composition is more general since, as indicated in Equation 5.1, languages can be seen as transducers emitting the corresponding subset of identity pairs and composition essentially becomes intersection.

deal with recognition and decoding. The most prominent and non exclusive (hence, compatible) cases where specialization is possible are: acyclic inputs (including strings and word graphs) and the use of some specific semirings.

5.7.1 Acyclic inputs

Acyclic inputs cover not only what is usually considered the input of traditional parsing algorithms (unweighted sentences) but also the extensions typically required for pattern recognition (several weighted input sequences expressed in a compact way by means of a word graph).

When composing an acyclic FSA with other not necessarily acyclic model, both of them without null transitions, the resulting model is also acyclic. In that case, the system of equations associated to these models, whose solution corresponds to a least fixed point, can be solved by using variable elimination. This also admits a recursive description which can be solved by means of dynamic programming (DP) [Bellman 1957; Cormen *et al.* 2009], as mentioned in Section 5.6. Besides, the entire procedure can be computed as the input model is being received. We will not have an in-depth discussion about the particularities of this approach since it will be explained in detail in Chapters 10, 11 and 12 where specialized decoders for acyclic inputs are described.

In particular, when the input model is a sequence, the composition is a multi-stage graph, which is a graph whose vertices can be partitioned so that these partitions form a sequence and edges always go from one set to the next one. DP algorithms, such as Viterbi, forward or backwards[91] algorithms use this kind of multi-stage graph that is commonly known as trellis where columns correspond to the stages of the multi-stage graph. The extension of these algorithms from sequences to DAGs leads to transitions between columns associated to transitions of the input DAG.

There may appear transitions between states associated to the same trellis column when using models with null transitions. There might appear loops due to these null transitions. It is possible, in this case, to resort to Dijkstra algorithm [Dijkstra 1959] when the semiring type fulfills certain conditions, as described in Section 5.6 and briefly discussed below. It is also possible to apply a variant of Bellman-Ford based on a previous topological sort, as described in Section 10.4.

The main difference between composition and these algorithms is that the first one constructs a model where semiring values label transitions whereas the other ones combine these values that, in the first case, could be aggregated afterwards by computing the shortest path on the resulting composition.

Relating the extension from FSA to RTNs of Section 5.5.3, it is also possible to apply DP algorithms either for the expansion and for the memorization approaches. The first case would be quite similar to re-

91 The difference between Viterbi and forward/backwards is often related to the type of semiring used: the Viterbi semiring in one case and the real semiring in the other cases. Viterbi can proceed in forward or in backwards traversal mode while this difference distinguishes forward and backwards algorithms which are often used to compute the posterior marginals and is used in the Baum-Welch estimation of parameters of HMMs.

cursive descent parsers whereas the second one is not much different of chart parsers. Indeed, we can establish some connections with Earley algorithm and the adaptation of our RTN composition to directly combine the resulting values in the case of acyclic input. This is just a rough comparison, not fine grained, since there is no equivalent to predictor, scanner and completer. We are just considering that both approaches are top-down tabular parsers compatible with processing the input in a predefined order.

We can observe that the memorization version described in Section 5.5.3 factorizes model suffixes and is based on maintaining a unique and implicit final state. A reasonable way to proceed is in setting is, perhaps, in backwards direction. Nevertheless, it is possible to describe RTNs and both expansion and memorization versions of Viterbi decoding using a different representation as the one proposed in Section 9.3.3 which has the particularity of specifying sub-models with an input point but, possibly, several output points. In the case of expansion, we could use an spaghetti stack to obtain a unique numerical identifier to each different sequence of model expansions and several models sharing the same sequence and the same entry point would share computations. In the case of memorization, the same numerical identifier is now associated to indicate that we have entered at a given submodel in a particular point of the input (a particular state of the input FSA). Again, models entering the same submodel but using different exit points might share resources. An example of such as model is a tree lexicon. The possibility of mixing expansion and memorization in the same model (using the best choice in a wiser way) is one of our future work assignments.

Cost of composition in terms of semiring operations

Since the cost of composition is related to the size of the resulting model, we will center our attention on it: when composing an acyclic input with a RTN model, both without null transitions and using the memorization approach, the number of states of the composed model may be quadratic with the number of states of the input and linear with the number of states of the model, while the number of transitions is, in the worst case, cubic with the number of input FSA states. These results coincide with the bounds of some classical parsing algorithms.

Let us observe that, when applying a shortest path over the composed model (or, more efficiently, when specializing a decoding algorithm to directly apply this computation) we have to measure also the cost of applying semiring operations that were only indicated in the composition. To this respect, there is a remarkable issue that we have not found elsewhere:[92] while the general description of this process has a polynomial cost in terms of basic semiring operations, it is widely known that optimal decoding is a NP-complete problem [Casacuberta and de la Higuera 1999]. Since this optimal decoding can be expressed by using a weighted language semiring, we can conclude that any implementation of this semiring must necessarily hide the gap between these costs so that the cost of these semiring operations grow with the increased complexity of the computed values.

92 Which does not mean that it has never remarked.

5.7.2 Semiring specific optimizations

Weighted context free grammars and the particular case of weighted finite state models are described as a sum of products. It is clear that some computation can be saved by using the distributive properties of semirings, but other possible savings can be envisaged for certain semiring types. A simple example with the Boolean semiring would clarify the exposition: we do not need to continue a summation once a true value has been obtained in a partial sum. Conversely, it is not required to pursue a sequence of products once the false value is reached.

parsing as search

The passage from arbitrary semirings to this particular case converts the sum of products into something more related to a search process where, now, it makes sense to try to figure out which paths would lead to the most higher values. Otherwise stated, most classical parsing algorithms simply does not make sense in a too general setting since they assume the use of semirings fulfilling some properties such as monotonicity and superiority (defined in Section 5.3). Superiority allows the use of both Dijkstra and $A^\star$ algorithms (the reader is referred to [Huang 2008a] for a tutorial).

5.8 SUMMARY AND SOME CONCLUSIONS

In this chapter we have reviewed most common formalism for some elements of the Chomsky-Schützenberger hierarchy, namely: finite state automata and context free grammars. We have also proposed some new algorithms. Most traditional works on parsing (usually centered on the huge plethora of parsing algorithms specialized in some restricted types of grammars suitable for compiler construction) start by defining formal models and parsing algorithms to process strings. These formalisms and algorithms can be later extended to deal with weights, outputs, retrieval of a packed shared forests or the capability to admit input data in the form of a (weighted) lattice.

unconventional review

Since we are interested in the use of parsing for recognition purposes and as a tool related to TSGMs, described in Chapter 4, we should focus on the weighted case, on parsing wight general CFGs (and not particular sub-families which are useful for compiler construction but not for natural language processing and other pattern recognition tasks) and, more importantly, when the input is no longer a sentence but a weighted lattice (a proper way to represent the ambiguities of the first stage of two-stage decoders). In this scenario, we consider that starting with classical descriptions in order to extend them later to the required extensions leads to unnecessary complexities and excessive nomenclature bloat as well as lots of confusing and avoidable concepts. One of the reasons is that parsing in these classical scenarios is mainly related to search whereas more general approaches deal rather with composing semiring values. That is why we have preferred a quite unconventional[93] approach that can be found in some papers but seems not yet the mainstream in our particular field.

93 Although we expect that it will become more and more usual.

semirings

We believe that it is better to start by describing, from the beginning, the parsing process as an operation on weighted languages and transducers not related to the actual way these languages are specified. The use of semirings has for long time proven its suitability for expressing different things in the same framework: acceptance, derivations, the set of best solutions, the most probable value in the Viterbi sense and many more. Despite the use of semirings, we have to be very cautious with the expression "semiring parsing" since it is a coined term [Goodman 1999] with an implicit connotation related to the use of deductive systems to describe the parsing process itself. We have used semirings from an slightly different point of view.

RTNs as normal form of CFGs

These languages and transducers, although are first presented in the form of grammars, are related later, for the case of context free models, with RTNs. Unlike most presentations of RTNs (usually plenty of procedural descriptions using sub-routines operating on a stack), we propose to view them as a particular *normal form* of CFGs (a restriction of the 2NF form).

parsing as composition

The parsing process itself is related to the "parsing as intersection" approach and also related to viewing CFGs as systems of equations. Some of these ideas can be traced back to [Bar-Hillel *et al.* 1961; Conway 1963; Ginsburg and Rice 1962; Lang 1988; 1994]. It is not clear if these approaches can be extended to other families of the Chomsky-Schützenberger hierarchy (as well as other families not neatly placed in this hierarchy), but we do not make any claim relating any of them. In particular, we are not dealing with synchronous CFGs sometimes used in SMT.

novel composition algorithm

Once the parsing as composition is defined and the proper formalism for defining both the input data and the model are described, we have to provide a composition algorithm. Instead of directly proposing such algorithm (and later justify its correctness, in the best of scenarios), we prefer to derive the composition of a FSA and a RTN from the definitions themselves following, to their ultimate consequences, the line consisting in viewing CFGs as systems of equations. In this way, and by manipulating systems of equations of input and models in order to obtain another system for the resulting composition, we are able to propose, for the first time (to the best of our knowledge), a composition algorithm that works fine with null transitions and semiring weights, even in the case of *non-idempotent* semirings, without resorting to the filter-composition trick/technique that seemed required in the alternative and superficially related "state-pair" approach to composition. Besides, the state-pair approach is often limited to the regular case (composition of two WFSTs) and not extended to CFGs. Some notable exceptions found in the literature are usually based on models represented by means of grammars [Bar-Hillel *et al.* 1961; Nederhof and Satta 2003; Dyer 2010; Hanneforth 2011] in some cases transformed into push-down automata but not into RTNs. In either case, a proper treatment of null transitions and semiring weights is not found in classical works.[94] A particularity, maybe a drawback, of our technique is the requirement to transform the input and the model to a

extended to CFGs by means of RTNs

94 The treatment of these null transitions together with the weighted case (often limited to the probabilistic case) can be found in more classical parsing algorithms related to input sequences as, for instance, in the probabilistic Earley parser of [Stolcke 1995].

normal form that we have coined "homogeneous epsilon". Constructive algorithms for this form has been proposed for the first time in this chapter.

homogeneous epsilon transformation

In our opinion, the approach used to derive the composition algorithm may contrast with algorithms given in the form of weighted logic deduction and, often, the use of hypergraphs. A common trade-off in science and engineering is to obtain the proper balance between generality, flexibility and simplicity. In this way, we are not against hypergraphs or against weighted deduction *per se*, we just want to highlight the fact that they may seem overkill in some cases and that a more general formalism is not inherently better in all circumstances. It is clear that hypergraphs are more powerful than our solution to the extent that they allow the use of arbitrary[95] n-ary weight functions besides the product of incoming values of hyperedge sources. However, the advantage of our approach is the possibility of further factorization of computations. In this way, hypergraphs seem overkill because they do not make obvious the additional factorization of computations which is obtained when the values combined in the hyperedge are simply the product of incoming values.[96] It is a pity that the research community has payed little attention to these advantages. Indeed, we have only found some notes on this issue in a quite recent work [Saers and Wu 2011].[97] Relating weighted logic deduction, we believe that the constructive approach proposed here provides more insight, although this may be a subjective opinion, a matter of taste.

hypergraphs can be overkill

Since algorithms described in this chapter do not need to resort to many concepts usually required in more traditional presentations (such as derivations, dotted-rules and the like), the exposition is more compact and economical. Just to point out another feature, the result of the parsing process is often represented, in most classical works, as a packed shared forest, while it is widely acknowledged that it is essentially equivalent to a CFG. In our proposal, RTNs are used for both input,[98] model and output.[99]

In our humble opinion, RTNs have been undervalued because most works dealing with them proposed algorithms that were cumbersome and procedural. We hope to be able to change this with the RTN-form and with the points of views and proposals of this chapter. Indeed, we would be quite happy if at least one result of this work is to promote and enhance the interest on RTNs. It would be a complete success if we convince at least *someone* to replace Earley or CYK by decoders based on the proposed RTN normal form and composition algorithms.

shameless lobbying RTNs

95 The use of monotonic functions is required for the correctness of certain algorithms [Knuth 1977] and dynamic programming approaches [Huang 2008b].

96 Partial products could be factorized when several hyperedges share the same prefix of tails.

97 It is not the central theme of [Saers and Wu 2011]. The central theme is the advantage of reifying the weight of rules of deductive systems by including them as additional entries in the hyperedges. In some point of the work, they describe hyperedges as mul/add sub-graphs and relate them with and/or graphs.

98 Taking into account that FSA constitute a particular case.

99 Some works use CFGs to represent the output [Dyer 2010]. Relating the use of RTNs instead of CFGs or packed shared forests to represent the output, [Dreyer and Marcu 2012] is one of the few exceptions proposing acyclic RTNs for this purpose. The possibility of using RTNs to represent packed shared forests is mentioned in [Iglesias *et al.* 2011] but, contrarily to our proposal, RTNs are described as a collection of FSA.

6 SEQUENCE/LANGUAGE MODELS

PREDICTION IS VERY DIFFICULT, ESPECIALLY ABOUT THE FUTURE.

Niels Bohr

CONTENTS

THIS chapter deals with the estimation of a-priori probabilities of word sequences, which is usually known as *language modeling*. The language model captures syntax, semantics and pragmatics of a language and provides the a-priori probability $p(w_1^N)$ of a given word sequence w_1^N. Note that these models are not necessarily related to what linguistics would consider a "language". LMs may rather model a domain specific language usually related to a particular task.

distributions over sequences

Models presented in this chapter are not limited to what is considered as "language models" by most recognition systems: probabilities over sequences can be found in other parts of recognition/decoding systems as illustrated in the following examples:

- the construction of a lexicon with multiple pronunciations usually requires to assign a probability to each one;

- some segment models such as the widely adopted HMMs (see Chapters 4 and 7) can be considered as the composition of a particular type of "language model" (which generates a sequence of types of emission probability models, taken from a finite repertoire) with a model explaining how the observed signal (divided into frames) has been emitted assuming these types of emission probability models.

Relating the relationship with the overall work described in this thesis, language modeling corresponds to the first stage of the "two stage generative model" (TSGM) proposed in Chapter 4. In that model, a sequence of units is generated in a first stage and a segment from the observed signal is associated to each unit in a monotone way.

statistical language modeling

Language modeling can be considered a particular case of weighted formal language formalism of Chapter 5 where weights are now interpreted as probabilities,[1] but this scarce description would lead to an incomplete view of this topic. For instance, since language models (LMs) are estimated from a finite training dataset, they probably have to deal with word sequences never encountered in the training data, which may require to apply smoothing techniques to assign nonzero probabilities to them. There exists other issues such as figures of merit, lexical units and specific models which justify to devote at least a chapter to LMs. Indeed, language modeling is covered in three different chapters in order to follow the division of this work into parts:[2] The current chapter addresses many LMs concepts from a more formal point of view, whereas implementation details will be delayed to Chapter 9 and, finally, some experimentation is delayed to Chapter 13.

We have been tempted to reduce this chapter to some bibliographic references to some of the many existing reviews but, as in other parts of this work, we prefer to provide our particular view which may seem less conventional than others found in the literature. For instance, most reviews are exclusively focused on count-based n-grams devoting a lot of nomenclature for each variant. Others talk about neural network LMs placing feedforward LMs closer to recurrent neural network LMs than to count based[3] n-gram LMs. We prefer to try to structure things as orthogonal as possible so that the fact of using connectionist techniques does not hide the fact of being n-grams.

The traditional way to introduce language modeling is either from the information theory/statistical point of view or from the formal grammar point of view (sequences are classified as either being grammatically correct or incorrect):

1 By selecting the proper type of semiring, but note that using certain semiring types is not enough to guarantee a proper and meaningful probabilistic interpretation without their correct manipulation, which is not always clear in the literature.

2 Each one corresponds to one of the points of the successful trilogy described in Chapter 1. Nevertheless, this division is not as perfect as we would like.

3 The widely known and statistical n-grams are cataloged as a particular case we have termed "count based" in order to distinguish them from other types of n-gram based, for instance, in feedforward connectionist models.

- from the information theory point of view, a sequence of symbols is emitted from an information source.[4] This approach allows us to express the fact that some grammatical sentences are more likely to occur than others, and also that some non grammatical sequences can, indeed, be observed. But note that some statements, such as *"all word sequences have the same probability"*, simply make non-sense in this framework since they do not lead to proper probability distributions;[5]
- the second approach, associated to formal language theory, is usually introduced in most reviews to be rapidly extended to the weighted case as described in Chapter 5. In this context, semirings can be used to describe, in a uniform way, different concepts such as probabilities, best derivations, the categorical assignment of a sequence to a language, etc.

It is reasonable to reconcile both approaches in the form of weighted formal grammars or other formalisms (described in Chapter 5) usually based on manipulating semiring values. Some methods, techniques and algorithms are more particular or may take profit of certain particular semiring properties. Note also that the use of certain semiring types is not enough to guarantee a proper and meaningful probabilistic interpretation without the proper and correct manipulation.

Chapter organization

The organization of this chapter is as follows:

- first, the most common probabilistic decompositions of the word sequence probability are discussed;
- another section addresses LM evaluation: the usual figure of merit to evaluate LMs, *perplexity*, is presented from the information theory point of view. Other measures, which try to take into account the observed segments used to represent these lexical units, are also reviewed;
- an overview of possible lexical units is presented to provide a more orthogonal view of language modeling: different types of lexical units can be combined with different types of LMs. This orthogonality makes unnecessary the use of specialized nomenclature associated to particular combinations of models and units.[6] This section covers the most common lexical unit types, the difference between open and closed lexicons, the problem of unknown words and the representation of lexicon. Lexicon representation may seem a trivial issue at a first glance, but it includes lexicalized models, distributed (continuous, factored,...) representations. Some lexicon representations try to alleviate the sparseness problem by making some lexical units from the training data

lexical units

4 Nevertheless, we do not consider streams but probabilistic models generating finite length sequences in a i.i.d. way.

5 Indeed, this claim violates $\lim_{N\to\infty} p(w_1^N) = 0$ generally applicable to any probabilistic LM described by p. It is possible to work, though, with improper distributions.

6 For instance, class based n-grams.

to contribute to adjust parameters used by other items which are "somewhat similar";

- other features which seem quite orthogonal to different types of language models are reviewed. In particular, concepts such as smoothing, combination of different LMs or the use of an external context associated to the wider problem "where decoding is just a part" (e.g. decoding a particular utterance in an automatic dialogue system can benefit from past history);
- the following sections are devoted to describe the classical LMs. They are mainly based on the prediction of the next lexical item given the previous history. The Markov approximation leads to the widely known n-gram models, but other types of LMs can also be described in this way. Statistical n-grams constitute a particular case that we have termed "count based" in order to distinguish them from other types. We have separated the management of the LM history and the set of features which can be obtained from this history. The manipulation of the LM history by means of LM identifiers makes it easier to use LM hypotheses in many types of decoder in a uniform way. The features used to estimate the probability of observing the next lexical item, given the previous history, constitute an extension of lattice-based LMs. This extension is proposed in Section 6.11;
- finally, other LMs issues are also discussed: how to deal with OOVs,[7], the LM look-ahead (see also Section 10.6.7), or the problems of using long-span LMs in decoding. These issues and other implementation details such as LM representations are described elsewhere into more detail, specially in Chapter 9.

6.1 PROBABILISTIC DECOMPOSITIONS

Although most works on language modeling are mainly constrained to the chain rule decomposition, disregarding other approaches, it is possible to classify language modeling by the different ways to decompose the probability of a sequence as follows:

- application of chain rule probability;
- whole sentence modeling;
- assigning values to different segments (spans of the sequence) and combining them.

Note that this classification is not exhaustive,[8] and note also that this division is not inherent to the proposed models but, rather, the most natural way to describe them. It is quite natural, for instance, to define

7 This topic has been covered in Section 6.3.1 but only to explain the special <UNK> symbol and the estimation of the probability of observing an OOV. Now we are concerned with the estimation of OOVs by mixing lexicon and lexicon-free approaches.

8 For instance, chain rule decomposition represents only one possible chain-rule ordering of the factors. There may be other types of LM which model the joint probability of sequences and other structured data such as trees, graphs, dependency graphs,... which could be removed by marginalization to obtain a distribution over sequences.

weighted context free grammars (CFG) by means of non-terminals and their associated production rules[9] which may seem more related to the combination of spans of a sequence to construct longer spans. It is also possible to obtain prefix probabilities[10] in CFGs [Jelinek and Lafferty 1991; Stolcke 1995; Nederhof and Satta 2003] which enables probabilistic prediction of possible following words as in the chain rule decomposition approach.

6.1.1 Chain rule decomposition and clustering of histories

Although we seek to estimate the probability of a whole sequence, it is usually desirable to be able to proceed from left to right applying the following chain rule decomposition:

$$P(w_1 \dots w_m) = \prod_{i=1,\dots,m} P(w_i | w_1 \dots w_{i-1})$$

based on the probabilities of observing each word given its prior context, which allows some decoders to proceed from left to right constructing longer and longer prefixes. Let us make a pair of remarks:

(a) relating nomenclature, the estimation of the first word is not the same as the estimation of a one word sequence $p(w_1)$, so the former is usually described as $p(w_1|w_0)$, where we can assume that w_0 is an special label <BOS> (begin of sentence):

$$P(w_1 \dots w_m) = \prod_{i=1,\dots,m} P(w_i | \text{<BOS>} w_1 \dots w_{i-1})$$

(b) it is also necessary to consider a distinguished label <EOS> (end of sentence) at the end of the sentence in order to make the sum of all finite length strings be 1. Without this trick, the sum of the probabilities of a given length would be 1 and the sum of the probabilities of all strings would be infinite since, as stated in [Chen and Goodman 1998; Page 4]:

$$\sum_w P(w | \text{<BOS>} w_1 \dots w_{i-1}) = 1$$

These special tokens, called context cues, allow us to distinguish between the probability of a given prefix p(<BOS> THE CAT IS SAD) and the probability of a complete sequence p(<BOS> THE CAT IS SAD <EOS>). This last probability can be computed as follows:

$$\begin{aligned} p(\text{<BOS> THE CAT IS SAD <EOS>}) = \; & p(\text{THE} \,|\, \text{<BOS>}) \\ & p(\text{CAT} \,|\, \text{<BOS> THE}) \\ & p(\text{IS} \,|\, \text{<BOS> THE CAT}) \\ & p(\text{SAD} \,|\, \text{<BOS> THE CAT IS}) \\ & p(\text{<EOS>} \,|\, \text{<BOS> THE CAT IS SAD}) \end{aligned}$$

9 Or by means of variables in an equation as was done in Chapter 5.

10 In general, the probability of a prefix can be computed by marginalization over of all sentences starting with the same prefix, although a brute force approach seems unfeasible. This is related to the approach known as "parsing as intersection" since the language of sentences sharing a common prefix can be easily represented by means of a FSM. This idea can be applied to compute infix and suffix probabilities [Nederhof and Satta 2003].

Markov approximation to clustering histories and n-grams

combinatorial explosion of histories

As can be observed, the number of different histories or prior contexts grows exponentially with their length. A way to restrain that combinatorial explosion (and one of the most successful approaches to construct LMs) consists of establishing equivalence classes over the set of histories:

$$P(w_i|w_1 \dots w_{i-1}) \simeq P(w_i|h(w_1 \dots w_{i-1}))$$

In particular, when this strategy corresponds to the Markov approximation, meaning that only the last $n-1$ words are taken into account:

$$P(w_i|w_1 \dots w_{i-1}) \simeq P(w_i|w_{i-(n-1)} \dots w_{i-1})$$

the resulting LMs are the widely known n-gram models[11] which will be covered into more detail in Sections 6.6.1 and 6.6.1. Note that other LMs based on the clustering of histories approach beyond n-grams are possible. When the number of histories is finite and $h(w_1 \dots w_i)$ can be computed from $h(w_1 \dots w_{i-1})$ and w_i, these LMs are regular, although the reciprocal is not always true.[12] In particular, it is widely known that n-gram models are essentially equivalent to a subclass of regular languages known as K-Testable Languages in the Strict Sense [García and Vidal 1990]. The use of regular LMs is essential in the approach based on composition of finite state transducers.

Other types of LM based on clustering histories do not assume a finite number of clusters of histories, as is the case of recurrent neural networks (RNNs) where the history is implicitly modeled with the internal network state (a continuous representation). They are discussed in Section 6.6.3.

6.1.2 Whole sentence LMs

Whole sentence language models [Rosenfeld 1997] consider the sentence *as a whole* instead of using the chain rule decomposition in order to proceed in a word by word basis. A set or bag of (usually binary) features is obtained from the sentence in order to compute its probability. These features are, quoting [Bellegarda 2004], *"arbitrary computable properties of the entire sentence"* which, in principle, could emulate the chain rule decomposition approach but, more generally, would include other features which cannot be taken into account in the previous case, such as additional features relating the context of the task where LMs are used, the sentence length, or the activation of certain triggers (e.g. the presence of certain words sequences). This makes it possible to capture longer distance relationships than (lower order) n-grams as well as more global features.

The probability of the entire sentence is obtained from those features following the maximum entropy approach, but note that the

11 An n-gram is a contiguous sequence of length n. An n-gram model is a LM based on n-gram counting and corresponds to a Markov model of order $n-1$. In practice, the term "model" is omitted and the LMs are just called n-grams. Some particular orders have special nomenclature such as unigrams, bigrams, trigrams, etc.

12 In some cases the history to compute the probabilities require more information to obtain the next history, as is the case of skip n-grams. This issue will be discussed later.

distinguishing feature of whole sentence LMs is not the use of exponential models. Indeed, there are other LMs called MELM[13] also based on exponential models and bags of features, although they use the chain rule decomposition (so they estimate $p(w|\text{history})$). On the other hand, whole sentence LMs estimate $p(\text{features}(w_1 \dots w_n))$. Exponential models guarantee that the sum of probabilities *over the space of features* has a total mass one. Unfortunately, in our humble opinion, this does not necessarily imply that the sum of probabilities over *the set of sentences* leads to mass one. The problem is that the function mapping sentences into features is not necessarily surjective (some feature combinations might not be possible) nor injective (several sentences may lead to the same set of features). Even worse, a particular combination of features may be obtained from an infinite set of sentences,[14] so that the distribution over the space of features cannot be translated to a proper distribution on the set of sentences.

Whole sentence exponential models are very expensive to train, and this is a serious drawback. Moreover, it is difficulty to use them in the same type of decoders as other LMs described by means of LM "states" (see Chapter 9), since a sentence is not completely known during decoding and most features cannot be computed. For this reason, they are better suited for multi-pass approaches to re-rank a list of candidates previously obtained with another type of LM.

Finally, let us remark that the possibility of describing a LM as a whole sentence LM does not imply that there exists other alternative ways to describe it. For instance, it is possible to describe a weighted CFG by means of a whole sentence LM based on features describing the presence of certain derivations in a CFG parsing.

6.1.3 Combining spans of the sequence

Weighted CFGs allows us to describe the probability of a sentence in terms of the probability of different spans of the sequence associated to models from a finite set (the set of non-terminals). Although CFGs can also proceed in a left-to-right way, the decoding procedure is usually different from the ones used with n-grams. Nevertheless, it is possible to apply the chain rule decomposition by computing prefix probabilities [Stolcke 1995], to adapt decoders from regular models to context free ones [Dupont 1993] or to approximate them by means of regular models [Mohri *et al.* 2001].

Weighted CFGs has been used as LMs for many years [Jurafsky *et al.* 1995] since they are able to model some linguistic phenomena better than regular ones although, in practice, regular LMs are more widespread nowadays for several reasons such as the possibility of using them in the weighted transducer approach,[15] the capability of probabilistic regular languages to approximate, with arbitrary precision, any stochastic language, and the good performance of n-grams.

13 Maximum Entropy LMs, succinctly described in Section 6.6.4.

14 For instance: using only features indicating the presence of a particular n-gram in the sentence. Note also that if a finite set of features has a finite number of values (as is the case of binary features) the set of feature combinations is finite.

15 Which allows to describe the decoding process in a uniform way and makes it possible the use of one step decoders.

6.2 ASSESMENT MEASURES FOR LMS

The evaluation of LMs is usually performed in two different ways:

INTRINSIC measures evaluates the LM *per se*. Perplexity is the most common figure of merit in this context. It measures the complexity of the language that our LM is trying to estimate. It also evaluates the inaccuracy of the LM w.r.t. the true distribution that is trying to emulate;

EXTRINSIC measures study how the LM behaves in an overall application. For instance, when we have to choose a LM to be used in an ASR task, we can try the different LM candidates on the overall system to measure the final performance on a validation set. The idea is to expect that this behavior could be extrapolated to test conditions. This is a realistic scenario not exempt from drawbacks: it is more expensive than other intrinsic measures,[16] it requires a task for testing and, perhaps more importantly, it is difficult to assess the contribution of the LM (this contribution might be diluted in or affected by other factors).

Extrinsic measures study the joint contribution of several models including those dealing with segment generation (acoustic models in ASR), the lexicon models (how lexical units are described by sequences of more elementary segment types) and the LM, but they also depend on the way these models are combined (decoding features such as different types of pruning to accelerate the search, the values of some parameters such as word insertion penalty, and so on).

Perplexity

Perplexity [Jelinek *et al.* 1977] is an information theory concept related to the notion of entropy. It measures the amount of non-redundant information per symbol conveyed, in average, by an information source, e.g. a source emitting a stream of symbols taken from a finite vocabulary. We are interested in probability distributions over finite sentences/strings instead of infinite streams.[17] Both cases are essentially the same when perplexity is normalized per word and when the emission of context cues is taken into account. This normalization is also related to the chain rule decomposition approach.

The entropy of a discrete random variable X measures the average information content required to know its value. It is defined as the mathematical expectation of the information associated to the probability distribution p associated to the random variable X:

$$H(X) = E(I(p)) = -\sum_{x} p(x) \log_b p(x)$$

16 Indeed, evaluating several components of the overall system in this way leads to a combinatorial explosion. There are some special cases such as the influence of grammar scale factor (GSF) and WIP (word insertion penalty) which can be solved by methods such as MERT [Och 2003]. Also, the statistical theory of design of experiments offers some clever ways to address the experimentation.

17 Some LM toolkits considered the set of sequences in a training corpus as a sole stream. The different sequences contained in the corpus were separated by context cue symbols, but this approach takes into account impossible n-grams composed by the final part of a sequence joined with the beginning of the following one.

where x ranges over the set of events of p. When measured in bits, the base 2 logarithm ($b = 2$) is taken and, in this case, the perplexity (PPL) is defined as $2^{H(X)}$. A notable example can provide us some insight about its interpretation: the perplexity of a uniform distribution over k discrete events (e.g. a fair die comprising k sides) is k. Otherwise stated, a perplexity of value k means that, in average, the information required to describe the next symbol is the same as a language with k equally probable symbols.

It is important to remark that what has been described is an inherent measure of the complexity of the task or the language which is given to us *as part of the specification of the problem*. What we seek is to measure the quality of a particular LM. Hence, we are in the realm of statistical machine learning since we try to approximate, by means of a model, an unknown probability distribution from a finite training set drawn from this distribution.[18] So, rather than the perplexity as defined before, we would like to measure the inaccuracy of a LM (with a distribution P_M) with respect the language we try to model, with unknown distribution P_L. Assuming that P_L is ergodic, the cross entropy between both distributions:

statistical machine learning

$$H(P_L, P_M) = -\sum_x P_L(x) \log P_M(x) \tag{6.1}$$

is an upper bound on the true entropy ($H(P_L) \leqslant H(P_L, P_M)$), which explains the interest in the *test set perplexity* as a figure of merit to evaluate LMs: assuming that we have an independent test set T at our disposal, the test set perplexity is defined as:

test set perplexity

$$PPL_M(T) = 2^{H(P_T, P_M)} = P_M(w_1, \ldots, w_n)^{-\frac{1}{n}} \tag{6.2}$$

where the lower is the better. In other words, we measure the predictive power of a LM by measuring how well it explains a set of independent sequences obtained with the same unknown distribution as the task we are trying to model. Training a LM to obtain the best test set perplexity leads to the widely known principle of maximum likelihood estimation of parameters.[19] It is also clear that test set PPL cannot be used to compare models coming from different languages or tasks. Also, even for the same task, the PPL measured on different test sets are not properly comparable. Another measure related with perplexity is the discrete perplexity [Van Den Bosch 2006; Sect. 2.4] which is suitable for measuring non-probabilistic text prediction systems, although in those cases it is possible to measure just the percentage of correctly predicted words.

18 We are assuming that this training set is drawn from the same distribution as the test, which is not always the case. There exists several techniques to perform model adaptation to deal with this mismatch. Training a LM can be seen as setting the parameters of a general model where from a given family (e.g. n-gram order).

19 There exist also an interest in discriminative techniques to estimate LM parameters by taking into account the set of correct and incorrect hypotheses produced by the overall system, e.g. [Kuo *et al.* 2002].

Segment generation confusability

perplexity is a coarse measure

Perplexity seems an appropriate figure of merit when we try to predict the next word given the previous one but, if we plan to use the LM as part of a recognition or of a translation system, we should consider the correlation between perplexity and the extrinsic figures of merit we would like to optimize (such as WER). This relationship has been investigated by several authors [Iyer *et al.* 1997; Chen *et al.* 1998; Clarkson and Robinson 1999; Klakow and Peters 2002] and it is widely known that, for instance, two LMs with the same test set perplexity can lead to different WERs.[20] Indeed, since perplexity is a coarse measure of goodness (the reciprocal of a geometrical average), two models with the same perplexity may assign different probabilities to the same particular sequence. Also, test set perplexity is measured on a finite independent validation partition, which is just an estimate. Perplexity does not take into account the confusability between the realizations of words (acoustic confusability in the case of ASR). Let us remark that the contribution of the LM is not so determinant to properly decode the less confusable words. If one model is better for less confusable words, other LM with the same perplexity can improve the WER if it helps better to deal with more confusable words, which is quite well explained in [Printz and Olsen 2002]:

> *the more distinct and unequivocal the pronunciations of the words in the recognizer vocabulary, the less important the language model. In the extreme, if every word sounded like itself alone, the language model would be irrelevant.*

Therefore, it seems more important, for decoding, to properly separate the correct hypothesis from the competing ones than to better estimate the likelihoods. Nevertheless, if one LM is significantly better than other in terms of perplexity, it would probably help to choose words better, including the confusing ones, and the corresponding WER should usually be improved.

Given those facts, it is not surprising to find alternatives to perplexity in the literature in order to obtain an analytic measure of the overall system.[21] The title of this section is our humble attempt to make "acoustic confusability" from [Printz and Olsen 2002] more task independent. The acoustic perplexity replaces, in the perplexity definition, the probability of the word sequence by the same probability *conditioned on the observed signal*. This concept is related to the information theoretic measures of conditional entropy and mutual information which, given two random variables X and Y, quantify, respectively:[22]

perplexity conditioned on observed signal

- the amount of information needed to describe the outcome of X provided that the value of X is already known

$$H(X|Y) = \sum_{x,y} p(x,y) \frac{\log_b p(y)}{\log_b p(x,y)}$$

20 Even a LM with a lower (better) perplexity can lead to a worse WER.

21 Since these concepts are more related to the overall system, they have been already discussed in Chapter 4.

22 See `http://en.wikipedia.org/wiki/Conditional_entropy` and `http://en.wikipedia.org/wiki/Mutual_information`.

- how much knowing one of these variables reduces our uncertainty about the other. They are related since:

$$I(X;Y) = H(X,Y) - H(X|Y) - H(Y|X)$$

Indeed, [Ferretti *et al.* 1989] proposes the use of the "Speech Decoder Entropy" defined as:

$$I(W;A) + I(W;C) - I(W;A,C)$$

and related to the loss of mutual information between the LM context and the word when the observation is also given. If perplexity is an estimate of the exponential of $H(W|C)$, the decoder entropy is obtained from $H(W|A,C)$. In other words, the combined information provided by the observed segment and the LM history together is less than the sum of their individual contributions because part of the information overlaps.

Unfortunately, the requirement of an observed signal associated to word sequences limits the applicability of this type of measures to validation sets where the observed signal is at our disposal. That is why some researchers have tried to estimate extrinsic measures (without the requirement of actual observed signals) by means of some type of the expected confusability between different segment realizations[23] and may even use some type of heuristics.[24] Let us remark that none of the proposed measures or studies we have found in the literature has considered the problem of detecting units associated to incorrect segmentations. For instance, in the case of HTR, the left part of the letter 'd' might be correctly classified as letter 'c'. We believe that just a confusion matrix between different types of label does not suffice and that the problem of extracting synthetic extrinsic measures, taking into account the joint segmentation and classification, is an interesting line of research which has not been fully explored.

6.3 LEXICAL UNITS

Words are commonly used as the basic lexical units in standard language modeling because they are the preferred output items in most recognition systems, but it is sometimes worth using other types of units in some cases. According to wikipedia[25], a lexical unit is

> *a single word or chain of words that forms the basic elements of a language's lexicon (vocabulary). Examples are "cat", "traffic light", "by-the-way", and "it's raining cats and dogs".*

This section reviews the use of different types of lexical units, other than words, which are commonly used in LMs. The use of categories is also discussed and, finally, the representation of lexical items is also

23 "Acoustic realizations" in the case of ASR.

24 For instance, in [Nanavati and Rajput 2006] the probability of occurrence of each word is divided into probability of recognition and probability of miss-recognition, but they assume that the later one increases with the probability of occurrence of the word.

25 http://en.wikipedia.org/wiki/Lexical_unit

considered, including factored models and continuous space representations. Note that the set of lexical units used by the LM and by the lexicon decoder usually coincide, but we are also interested in studying the possibility of decoupling them so that they could be different.[26] Note also that some LMs described in the literature, such as the hybrid models [Choueiter 2009; Section 2.2.2], combine different lexical unit types in the same model.

6.3.1 Open vs closed lexicons

When constructing a LM, a lexicon is usually obtained by taking the lexical items encountered in a given corpus or cataloged in a dictionary. Closed vocabulary tasks assume that no lexical item outside this lexicon can appear in test conditions. On the other hand, open vocabulary tasks are those where the lexicon *cannot be bounded a priori*. Since the lexicon is usually obtained from a finite training set, how can this be reconciled with open vocabulary tasks? Without the proper mechanisms, words out of the closed vocabulary, also known as Out Of Vocabulary (OOV) words, are sure errors.[27] Trying to avoid the problem of OOVs by augmenting the lexicon may alleviate the effects since the frequency of words follows a power law distribution.[28] Indeed, it can be even desirable to ignore less frequent lexical items from the training data by means of a *count cutoff* threshold, meaning that all items whose frequency is below this threshold are replaced by a special token denoted by <UNK> to represent unknown words. The presence of the <UNK> token in the training data allows the estimation of the probability of encountering an unknown word[29] given the previous LM history. The actual use of these LM transitions depends on the capability of decoders to detect OOV words, an issue that will be addressed in Sections 6.8 and 10.7. Note also that, in principle, the <UNK> token could be divided into several classes of OOV words as far as the rest of the decoder is able to properly use them.

6.3.2 Most common types of lexical unit

The following list reviews the most common types of lexical units. Context cues and OOVs are not present in this enumeration since they can be used with nearly any lexical unit type in an orthogonal way. This list is biased towards linguistic applications, but note that there can be other lexical unit types which can take the same role in decoding.[30] For instance, some bioinformatics applications use language

26 Many particular cases can be handled easily in the weighted FST paradigm when the conversion between them can be performed by means of transductions, the use of dynamic decoders could require the use of some wrappers to be described later.

27 Besides, OOVs are often content words and the associated decoding errors often propagate to neighboring words during decoding.

28 Known as Zipf's law. See, for instance, `http://en.wikipedia.org/wiki/Zipf%27s_law`.

29 This token thus represent a category of group of words so that its probability should be distributed into the possible unknown word hypothesis.

30 The transducer composition approach makes unnecessary to distinguish the place in decoders and the concept of lexical unit is just one of symbol alphabets. We refer here to the fact that they act as the first stage in a two stage generative model.

models where lexical items are base pairs.[31] Let us finish with another example: the generation of frame sequences in the second stage of the generative process described in Chapter 4 is usually modeled by means of HMMs, which can be considered as the composition of a LM (generating a sequence of "probability density functions" or pdf's) with models describing the likelihood of emitting the observed frames given those types of pdf's. From this point of view, "type of pdf" would be another type of lexical unit.

Words

Words are defined as the smallest form that may be uttered in isolation and which have some content or meaning. They may consists of a single or several morphemes and are usually associated to white-space separation in many writing systems. They are, not surprisingly, the most classical and widespread type of lexical unit used in many tasks related to the problems we are studying.

Sub-words (inflected languages and lexicon-free approach)

Sub-word units cover from phones/graphemes to morphemes, including syllables and even short phone sequences obtained automatically. There exists at least two main applications of LMs based on sub-word units: on the one side, to construct lexicon free recognizers which are, in principle, able to deal with OOV words (see also Section 6.3.1) or to proceed in a lexicon independent manner in order to decode words afterwards [Whittaker *et al.* 2001]. On the other side, to construct LMs for inflected languages which are based on morph composition [Hirsimäki *et al.* 2006] since, for those languages, the lexicon can be considered unlimited at the word level.

Note that sub-word based LMs based on n-grams take into account the last $n-1$ sub-word units, making it possible to roughly model words when using a sufficiently high n-gram order. In this way, lexicon free approaches are able to somewhat learn a lexicon and, at the same time, to be able to detect OOV words (see Section 6.8).[32]

Pairs (words,concept), bilingual tuples

The use of LMs estimating the probability of sequences of word-concept pairs or bilingual tuples is related to weighted transducers used in some translation systems and in some SLU systems (to perform concept tagging which can be seen as a particular case of monotone translation, see Section 2.4). Some works, such as [Bayer and Riccardi 2012], show that the best LM for recognition are not necessarily the best ones for SLU. These types of lexical units can be represented by means of factored LMs described in Section 6.3.4.

31 According to `http://en.wikipedia.org/wiki/Base_pairs`, they are defined as the linking between two nitrogenous bases on opposite complementary DNA or certain types of RNA strands that are connected via hydrogen bonds.

32 Some works related to the lexicon-free approach are described in the experimental part in Chapter 15.

Multi-word expressions, compound words, phrases,...

There exists many different types of multi-word expressions (see [Sag *et al.* 2002] for a review). Phrases are word sequences which are recurrent in constrained domain languages and can be thought as a single lexical entry. Within the area of corpus linguistics, collocation defines a sequence of words or terms that co-occur more often than would be expected by chance.[33]

The fact of considering those multi-word expressions as a type of lexical item fall into the "words-with-spaces" approach, which seems the most immediate way (but not the only one) to deal with them. Indeed, this approach may require to explicitly enumerate a large number of variations leading to a problem known as lexical proliferation which is translated, in the case of decoding, to an increase of the lexicon size.

Compound words can be used to model some word contractions and to model the phenomenon known as *liaison*. For example, in [Finke and Waibel 1997; Section 3]:

> *Ignoring the word neighbors and still allowing for all sorts of phonetic reduction would result in a long list of confusion pairs of very frequent words. Consider of example word sequences like "KIND OF" and "SORT OF" which are often reduced to "KINDA" and "SORTA".*

but note that this issue seems more related to the estimation of the lexicon (discussed in Chapter 10) than to the LM, depending on the capability of the decoder to use a lexicon and a LM using word sequences. A phenomenon similar to liason is the use of across-word context dependent units,[34] which increases the complexity of decoding.

alleviate lack of across-word context dependency

The use of multi-words can alleviate the lack of across-word context dependency when using context dependent units (i.e. when only within-word context dependent units are used). Another advantage of using multi-words, from the decoding point of view, is the observation that compound words are usually less confusable acoustically (in ASR systems). That is: recognition mistakes occur, generally speaking, less frequently in longer words. For those reasons, compound words can be automatically derived for decoding purposes, usually by selecting word groups which appear more frequently [Saon and Padmanabhan 2001].

The use of multi-words is not restricted to sequence recognition: phrase based LMs are widely used in translation either as part of bilingual tuples or to model the target language. The lexical proliferation associated to the use of multi-word expressions makes the problem of data sparseness even more acute and, not surprisingly, the combination of phrase based LMs with categories (see next point) has been recently proposed [Justo-Blanco 2009].

33 As defined in `http://en.wikipedia.org/wiki/Collocation`.

34 Indeed, the liason phenomenon can also be automatically tackled using the same techniques, see Section 10.2.4.

6.3.3 Categories

In this context, categories usually refers to groups of words, they are also called word classes or just classes. The main motivation for using categories, in the context of LMs, is to reduce the size of the LM and hopefully, with less parameters, to improve its estimation. To this end, it is important to group those entries that exhibit a similar behavior in the LM.

It is tempting to include them in the previous list of lexical units because it is possible to design a LM over sequences of categories, but we believe they deserve a different section for two reasons: On the one side, they are orthogonal of the lexical unit. It is possible to use categories of words, sub-words, phrases, etc. On the other side, it is possible to use a LM of categories to obtain a LM over other lexical units imitating the two stage generative problem where first the categorized sequence is generated[35] and each label is replaced by a lexical unit of the second LM afterwards.

When a word may belong to several categories, it is necessary to take into account several categorized sequences, which can be efficiently performed by means of dynamic programming using the same ideas as other two stage generative sequence tasks. But this is not the only way to use categories: it is also possible to use categories only when predicting the word and not in the previous history, or to use different categorizations in different positions of the predicted word and on the specific positions of the past history [Yamamoto *et al.* 2001], etc. When the categorization is applied over the predicted word, the expansion of the category can also take into account the LM history as follows:

LM history dependent expansion

$$p(w|h) = \sum_c p(w|c,h)p(c|h)$$

where the summation can be skipped when each word only belongs to a unique category.

Different categorizations can be used to obtain alternative LMs to be combined afterwards (see Section 6.4), a novel way to deal with categories is as follows:[36]

factored categorizations

$$p(w|h) = \sum_{c_1 \in C_1, c_2 \in C_2} p(w|c_1,c_2,h)p(c_1,c_2|h)$$

which, assuming certain independence assumptions, can be used to factorize the output of a LM into the combination of smaller classifiers.

Most category-based LMs ignore the dependency on the LM history, but it is useful, for instance, in the "shortlist approach". In this approach, associated to neural network n-grams (NN LMs), a subset of the lexicon, known as "short list", usually composed by the most frequent words, is modeled without categories whereas the rest of the lexicon can be grouped into a sole category known as "Out-Of-Shortlist" (OOS), represented by the <oos> token, as follows:

shortlist approach

35 And it is a common mistake to measure the perplexity at the category level ignoring the last term unless we were evaluating the generation of sequences of categories.

36 It is described for two categorizations but can be generalized in a straightforward way.

$$p(w|h) = \begin{cases} p(w|\text{<oos>}, h)p(\text{<oos>}|h) & w \in \text{OOS} \\ p(w|h) & \text{otherwise} \end{cases}$$

where the expansion of the <oos> token can use another auxiliary language model p':

$$p(w|\text{<oos>}, h) = \frac{p'(w|h)}{\sum_{w' \in \text{OOV}} p'(w'|h)}$$

Observe that the summation of the denominator could be pre-computed and taken into account in a LM representation. This and other techniques will be discussed when dealing with NN LMs.

Relating the construction of categories:

- it can be related to the task (e.g. grouping cities in a CITY category for a train reservation dialogue task);
- by applying linguistic knowledge associated to the language (e.g. converting words to their root form or stemming); or
- by means of automatic clustering techniques (agglomerative, iterative by exchanging,...) which try to maximize the likelihood of the training data or optimize the perplexity of the resulting LM.

Decoders can deal with categories in different ways. In a transducer composition approach this could lead to an expansion, for each transition labeled with a category, of all words associated to it. In a dynamic decoding approach, they can be entirely handled by the LM making this issue transparent to the rest of the decoder (and even for the underlying categorized LM) as described in Section 6.9. The lexicon representation, discussed in the next section, is also related to categories since we can categorize certain factors of a factored representation and because some techniques for obtaining continuous and structured representations may assign similar representations to certain words.

6.3.4 Lexicon representation

The description of problem types related to sequences described in Section 3.1 introduced a well known classification of types of random variables. According to this classification, words could be described as symbolic or categorical since, at a first glance, "they are finite, discrete and have neither a notion or order nor distance". From this point of view a word, in a LM, is modeled by means of an arbitrary identifier and the contents of this section would be trivial. But this is not completely true since a distance or similarity between words can be established and some LMs can take profit of it making words in the training data to contribute to the estimation of similar words. This can be achieved by replacing the conventional flat representations by others with some type of structure. Quoting [Turian *et al.* 2010]:

flat vs structured representations

> *A word representation is a mathematical object associated with each word, often a vector. Each dimension's value corresponds to a feature and might even have a semantic or grammatical interpretation, so we call it a word feature.*

It is possible, for instance, to embed word representations into a space in order to use the distance in this space as a measure of similarity between words. Depending on how this embedding is obtained, this similarity may be associated to the LM or can also take into account semantics or even the similarity between their observed realizations (e.g. acoustic similarity in the case of ASR), as described in Section 4.4.5 on representation learning.

The decomposition of lexical units into factors leads to factored LMs. Some of the factors can also correspond to continuous representations, making it possible to combine different types of representation in the same model. Also, since a model with one factor is a (trivial) factored model, this representation extends the previous ones. Indeed, the representation of lexical items can go beyond a "bundle of features" and include some structure which can be manipulated by means of operations such as union or intersection (e.g. [Blanc 2006; Section 2.5]) in order to be used with some formal grammar and automata operations adapted from simple alphabets to this structured representation. Although the following division between continuous and factored representation is not completely disjoint (since they can be combined), not any factored representation is a continuous one.

Continuous representations

Continuous representations, as opposed to symbolic or categorical, emphasize the idea of having a notion of distance between words which can be used by LMs to improve smoothing, i.e. to better model the estimation of a word, given the previous history, when this particular combination has not been observed during training. A usual smoothing approach consists of discounting some probability mass from observed events in order to assign it to unseen events. This is achieved by interpolating with or by backing-off to simpler models (e.g. lower order n-grams), but a continuous representation makes it possible to rely on similar words and word histories, even for lexical items which have been never observed.

Following [Turian *et al.* 2010], word representations can be roughly divided as follows:

- in a distributional representation, the vector representation is obtained from the occurrences of the word in different contexts of the training data, which is usually represented by means of a co-occurrence matrix. The rationale behind this approach is the *distributional hypothesis* which states that words appearing in similar contexts have probably similar meanings or behave similarly from the LM point of view [Turney *et al.* 2010]. This representation is usually compacted by means of techniques such as the truncated singular value decomposition and is related to latent semantical analysis (LSA);

- a common way to obtain a continuous representation based on the LM performance consists of using a projection layer in neural network language models (NNLMs, described in Section 6.6.1) where the codification of lexical units is implicitly learned as part of the LM training [Bengio *et al.* 2003]. In this way, similar words

are probably assigned nearby codes. Other works such as [Maas *et al.* 2012] try to map entire words from speech signal making it possible, in principle, to obtain a continuous representation which takes into account both the semantics and the acoustics (can be generalized to other tasks), which seems a desirable property to improve decoding. This is also related to representation learning described in Section 4.4.5.

Among the previous advantages of continuous representations for language modeling to improve smoothing, estimation and even for automatically generating categories, we can also mention the possibility of better handling OOVs: [Jelinek *et al.* 1990] proposed a technique to deal with OOVs by associating them to the existing lexicon entry whose context was better adapted to their situation. Continuous representations could offer a greater flexibility to apply this idea since several words satisfying this constraint can be taken into account and the obtained code would be no longer a synonym. The use of techniques mapping words from the observed signal [Maas *et al.* 2012] would lead to even more direct ways to deal with OOVs.

Factored LMs

According to [Bilmes and Kirchhoff 2003], factored LMs are those where each lexical items is represented by means of a vector of k factors. These factors can represent different things such as root forms, morphological classes, information such as gender and number, categories, etc. As an example of the advantage of the factored representation, let us imagine how easy it is to model gender and number agreement (so common in many natural languages) when gender and number are part of the factors.

Relating the advantages of this representation, some factored LMs may be described as a combination of simpler models, each one based on a different factor subset. Smoothing techniques can take profit of this feature to generalize the classical concept of back-off used in n-grams (e.g. parallel back-off [Bilmes and Kirchhoff 2003]). This generalization will be further developed in Section 6.11.

The factored representation is more suited for some LM types than for others. Factors fit more naturally in some LM types such as feature-based n-grams (being NN LMs the most notable case) or Maximum Entropy LMs than in count based n-grams (detailed below).

related to factorial HMMs

We can find an analogy between factored LM and factorial HMMs [Ghahramani and Jordan 1997] (see also Section 4.4.1). Indeed, HMMs can be seen as the composition of a LM emitting sequences of types of emission probabilities (pdf's) and the corresponding model emitting the observed frames. On the other side, factorial HMMs are HMMs with a distributed representation for the hidden states.[37] In this way, the use of a factored LM in the generation of types of frame emissions in HMMs would naturally lead to "factorial HMMs".

37 Each state in the distributed representation of Factorial HMMs *"evolves according to its own dynamics"* meaning that its value in a given position only depends on the same state variable in previous states, which is a quite strong independence assumption with respect the most general case.

6.4 LM COMBINATION

Since LMs are, in essence, probabilistic classifiers, LM combination is just a particular case of combination of classifiers. When using whole sentence LMs, their combination should be done at the sentence level but, when models to be combined are based in the chain rule decomposition and are using the same lexical unit types, this combination can be applied when predicting a lexical unit given the prefix or LM history. There are several reasons for combining LMs:

- in some cases the training data comes from different sources and have quite different sizes. It is possible to train a separate LM from each source and to combine them later adapting the weight or importance of each LM w.r.t. the current task, usually employing a separate held-out dataset;
- in other cases, the corpus is composed of different types of sentences (not necessarily from different sources) and it is possible to assign each one to a particular topic and to train a topic dependent LM. During recognition, LMs are combined depending on the probability of each topic for the current sentence;
- some LM types seem more reliable depending on the LM history. For instance, depending on the count of n-grams related to this LM history. In this case, the combination of LM may try to boost the most reliable model for each situation. In general, LM combination can be used to implement smoothing techniques, specially for combining higher order with lower order n-grams;
- LM combination can also be used to perform model adaptation (see Section 6.5). A typical way to implement this feature consists of estimating a task specific LM with the limited task data (for instance, a cache unigram [Kuhn and De Mori 1990]) which is combined with a more general or "background" model;
- some LM types have scalability problems w.r.t. the lexicon size, (other LM types can also benefit from this idea since, for instance, the use of categories may reduce the memory footprint and improve estimation of parameters). A common trick to use these models on large lexicons consists of grouping less frequent words in categories and to use another LM to estimate the probability of a word given the category *and* the LM history. An example, described below, is the use of a shortlist at the output of neural network LMs, but this idea can be used as a general technique as described in Section 6.11.

Given n LMs based on the chain rule decomposition modeled by their respective probabilities $p_i(w|h)\, 1 \leqslant i \leqslant n$, their combination is usually performed in two different ways:

- linear combination of components:

$$p(w|h) = \sum_{i=1}^{n} \lambda_i(h) p_i(w|h)$$

  where the interpolation weights $\lambda_i(h)$ satisfy the following constraints: $0 \leqslant \lambda_i(h) \leqslant 1 \,\forall\, h$, $1 \leqslant i \leqslant n$ and $\sum_{i=1}^{n} \lambda_i(h) = 1 \,\forall\, h$ and are usually tuned to optimize the set perplexity on validation;

- log-linear combination of LMs:

$$p(w|h) = \frac{1}{Z(h)} \exp\left(\sum_{i=1}^{n} \lambda_i(h) \ln p_i(w|h)\right)$$

Interpolation weights are functions depending on the LM history but a simplification or particular case is the use of constants. Note that log-linear and linear weights have different interpretations and let us also remark that a normalization term $Z(h)$ is required in the log-linear combination to yield a probability distribution.

linear interpolation cannot hurt

Linear interpolation is usually seen as a union of experts meaning that the result is usually broader than each individual component and, indeed, an optimally interpolated model is not worse than any of its individual components. Contrarily, the log-linear combination penalizes the disagreement between the different LM constituents to the point that if one of them estimates a zero probability the overall result becomes zero.

Linear and log-linear interpolation can be combined [Liu *et al.* 2013; Sect. 2.2]. There exists other combination methods, as is the case of using $p_i(w|h)$ as inputs for an exponential model. Combination can also be performed at the parameter level, meaning that instead of combining the output probabilities of the different LMs, we combine the parameters required to construct the model, which requires that all LM constituents must be of certain type. An example is the technique known as count merging [Bacchiani *et al.* 2006] where n-gram counts are combined in the case of count based n-grams.

combining different LM histories

Relating the way LMs are actually performed, in some cases the LM combination can be pre-computed and even represented in the same formalism as the LM constituents, as is the case of n-grams. Also NNLMs (described below) can be combined in an ensemble but can also be *distilled*[38] in a new model [Hinton *et al.* 2014]. In the general case, when LMs are of heterogeneous nature, LM combination can be performed on the fly. For instance, it is possible, when LMs implements a particular LM interface, to construct a LM wrapper which implements the same interface while the LMs from the combination do not need to be aware of this use and remain unchanged. LM interfaces are detailed in Section 9.1.

When combining LMs based on the chain rule decomposition and clustering of histories approach, the LM history clusters in each model may differ and this poses some implementation problems. It would be convenient to distinguish between the minimum LM history information required to obtain the next history with the next lexical unit and the information obtained from this LM history required to actually compute the probability of the following lexical unit. This distinction is usually overlooked since in most common LM types, as is the case of n-grams, they coincide and, although this is relevant for LM combination, we prefer to delay this discussion to Section 6.11 where it will be discussed in a more general context.

38 Training the new model with soft outputs and using a higher softmax temperature, hence the name distilling.

6.5 DYNAMIC CAPABILITIES

adaptation from context

Context greatly influences which sentences are more likely to be produced. Recognition systems such as broadcast news must cover a wider range of topics than other more semantic restricted systems like a hotel reservation dialogue system. But, in many situations, topics do not change very fast and, in these cases, it may be useful to detect the past topics in order to boost the sentences which are more likely related to them [Martin *et al.* 1997; Mahajan *et al.* 1999; Bellegarda 2004; Steyvers and Griffiths 2007]. This capability of adaptation may be obtained in several ways.

It is important to distinguish the capability of adaptation *during decoding* from the adaptation of a general LM to a domain by means of a specific corpus. In both cases, we can talk about "adaptation data" but, in the dynamic case, we may have more available data at our disposal. We can distinguish the following sources of adaptation data:

- information from past utterances;
- a guess of the current sentence we are decoding;
- other type of information from the past and current sentence besides the textual content;
- information from the external context which, in general, cover heterogeneous information related to the task.

external context

Besides the information obtained from previous sequences, in some cases there may be an external context of heterogeneous nature which could be taken into account: for instance, in a spoken telephone dialogue system for hotel reservations we could consider the geographical position of the caller[39] or the current date which could help to better guess the probability of looking for holiday destinations, winter resorts, etc. An example of information from past and current sentence besides the textual content is the estimation of the speaking rate in speech.[40] It has been shown that speaking rate may influence the use of certain words [Fosler-Lussier and Morgan 1999; Ward *et al.* 2012].

Note that the prefix of the sentence we are currently decoding (where the LM is being used) is *not* included in what we understand by adaptation data but this is what has been called "LM history". The LM history is not usually employed to dynamically adapt the LM for a quite obvious reason: this would break, in most cases, the clustering of histories approach making all LM histories to become different and, hence, this would lead to a unaffordable decoding process. Using the whole current sentence seems even worse to this respect and, ultimately, it poses the same problems for decoding as the use of whole sentence LMs. Fortunately, there is an alternative: to use information extracted in a first decoding pass.

suited for multipass decoding

The problem of using from previous sentences or from a first recognition of the current sentence is twofold: on the one side, the available data to perform this adaptation is too scarce, on the other side,

39 It is not always possible to obtain the caller location, so this value should be optional in any case. Note that this information can also be used to adapt lexicon and acoustic models to deal with regional accents.

40 It can be obtained by external measures of from a previous recognition stage. This issue has already been discussed in Section 2.9.3.

risk of reinforcing mistakes

these sentences usually come from an automatic recognition system and recognition errors could result in a reinforcement of mistakes, a kind of positive feedback.[41] Information from past utterances can be used at least in two different ways described below.

Topics

A set of topic-dependent LMs can be estimated after classifying the training data sentences into those topics. It is usual to train a general or "background" LM with all the available data. During decoding, adaptation data is used to estimate the probability of each topic in order to weight each topic-dependent LM as well as the background LM. There exists several techniques for clustering the corpus into topics and for tracking the topics from the adaptation data, some of them based on co-occurrences of words.

Note also that it may be interesting in some cases to proceed to decode the same sentence with different topic-dependent LM in order to provide different semantic interpretations instead of combining the LMs during decoding.[42] Note also that, besides a weighted combination of topic-dependent LMs, we can also train a sole LM but to provide the estimated probability of being in each topic as an additional input of the statistical model, which seems trivial to implement, for instance, in NN LMs.

Cache-based LMs

Cache-based LMs where introduced in [Kuhn and De Mori 1990] where some words were considered to be self-triggered (recently seen words were considered to be more probably observed). The term cache is generalized to trigger-triggered pairs and, in general, to extract a summary of recent sentences as additional features for dynamic LMs:

- a cache of the most recently seen n-grams which is smoothed together (typically by linear interpolation) with the static model [Kuhn and De Mori 1990; Jelinek *et al.* 1991];
- some word from past context boost the probability of words that they trigger. Trigger and triggered form pairs of mutually informative words [Rosenfeld 1996];
- context is used as additional parameters in a n-gram based LM. For instance, [Zamora-Martinez *et al.* 2012] proposes an extension of NNLMs to take into account this cache information which represents the concept of bag of words combined with an exponential decay in order to (partially) take into account the order, distance and number of word (and, optionally, semantic concept) occurrences.

41 In some tasks such as in automatic spoken dialogue systems, the dialogue manager can adopt measures to validate some hypothesis and these confidence measures could be used to improve dynamic LMs, which is an interesting feature that we have not found in the literature. In general, recognition systems can provide not just the best hypothesis but the set of most promising ones (in form of word graphs, etc.) together with confidence measures and this has been used in some LM adaptation techniques.

42 This is another interesting decoding problem discussed elsewhere in this writing: how to reuse parts of the computational effort.

6.6 COMMON LM TYPES

It would be unfeasible to make an exhaustive enumeration of all existing LM types. We seek to structure a subset of these models as a combination of basic ideas to justify the proposal of Section 6.11. The majority of LMs types found in the literature follows the chain rule decomposition and the clustering of histories approach.

6.6.1 N-gram models

The dominant language modeling technology, n-grams, uses the Markov assumption or approximation to perform this clustering. A kth-order Markov model is a dynamical system whose next state only depends on its k most recent states. An n-gram is a contiguous sequence of length n. An n-gram model is a LM based on n-gram counting and corresponds to a Markov model of order $n-1$:

surprisingly difficult to improve on

$$h(w_1 \dots w_{i-1}) = w_{i-n+1} \dots w_{i-1}$$

In an abuse of notation, the term "model" is sometimes omitted and the LMs are just called n-grams. Some particular orders have special nomenclature such as unigrams, bigrams, trigrams, etc. These models are very well suited to capture local interactions and a higher order n-gram is able to capture longer dependencies and is, therefore, more powerful in principle, that its lower order counterparts. Unfortunately, increasing the order poses some problems not only in terms of estimation and model size (specially in count based n-grams), but also on the number of different LM histories which consequently increases the cost of decoding.

The number of different n-grams is $|\Omega|^n$ being Ω the lexicon and $|\Omega|$ its size. And that is why, excepting trivial cases, the training material is always sparse (due to the curse of dimensionality).

count based vs feature based

We can distinguish two main approaches to n-gram modeling: the estimation of parameters based on counting n-grams in a training corpus and the more general technique based on estimating the parameters of a continuous statistical model which computes the probability of the following lexical item from features extracted from the last $n-1$ ones. The first approach is closely related to weighted automata and transducers. Indeed, n-gram models are often converted to these forms. Although any n-gram model is a particular case of a weighted finite state model (essentially equivalent to a subclass of regular languages known as K-Testable Languages in the strict sense [García and Vidal 1990]), it does not seem worthy to pre-compute these automata for n-grams based on other type statistical models such as multilayer perceptrons (MLPs), since this would lead to an unnecessary increase of the model size.[43] It is also important to remark that many models based on features are not necessarily limited to the n-gram assumption unless we obtain their inputs from the last $n-1$ last lexical items and that many models based on finite state techniques are not n-grams.

43 Nevertheless, there exists several proposals [Arisoy *et al.* 2013; Wang *et al.* 2013]. Even for count based n-grams, the conversion into automata has to deal explicitly with back-off to avoid an exponential increase of its size, this will be discussed into more detail in Chapter 9.

Count based N-gram models

Count based n-grams are the particular case of the maximum likelihood estimation of the probability $p(w \mid h)$ of observing a word w given its history h when the history follows the n-gram approximation. In this case, these probabilities correspond to the relative frequencies observed in a training corpus after applying smoothing techniques to deal with the unavoidable problem of data sparseness. Without those smoothing techniques:

$$p(w \mid h) = \frac{N(h, w)}{N(h)}$$

where $N(\cdot)$ denotes the number of times an event has been observed in the training data. For instance, in the particular case of a trigram:

$$p(w_3 \mid w_1, w_2) = \frac{N(w_1, w_2, w_3)}{N(w_1, w_2)}$$

smoothing

There exists a plethora of smoothing techniques to solve the problem of data sparseness which may explain why some events with a nonzero probability may have not been observed in the training data. Our objective is not to explain all the available smoothing techniques, since there already exists many good reviews such as [Chen and Goodman 1998; Section 2] or [Gutkin 2000; Chapter 2], to mention a few. Let us just sketch the very basic ideas: many smoothing techniques try to share a part of the probability mass usually assigned to the observed events with non-observed ones.

One of the oldest approach known as additive (a.k.a. Laplace or Lidstone) smoothing consists of adding a positive value $0 < \delta \leqslant 1$ to every n-gram count and to properly normalize afterwards. An approach known as linear discounting scales the maximum likelihood estimates by a constant slightly less than one to have some known probability mass at our disposal, which can be then redistributed. Another technique to obtain some probability mass from observed events is absolute discounting where a little constant value is removed from observed n-gram counts. Some smoothing techniques make use of the Good-Turing frequency estimator[44] or similar ideas based on cross-validation techniques to correct the n-gram counts based on the number of times they have been observed in different corpus partitions.

back-off

Relating how to deal with non observed events, a general technique consists of using lower order n-grams either by means of a linear interpolation of by consulting in a given order from most specific to less specific ones as with back-off [Katz 1987]. According to [Kneser and Ney 1995], most smoothing techniques can be described by means of the following back-off equation:

$$p(w|h) = \begin{cases} \alpha(w|h) & N(h, w) > 0 \\ \gamma(h) Q(w|\hat{h}), & N(h, w) = 0 \end{cases}$$

where $\alpha(w|h)$ is a reliable estimation of $p(w|h)$, $Q(w|\hat{h})$ is an estimation for a less specific (lower order) history and $\gamma(h)$ is a scaling factor

44 See http://en.wikipedia.org/wiki/Good%E2%80%93Turing_frequency_estimation.

obtained from $\gamma(\cdot)$ and $Q(\cdot)$ in order to be a probabilistic distribution. Is is also possible to model a linear interpolation in this form, which explains the success of the back-off based n-gram formats.

The idea behind Kneser-Ney smoothing [Kneser and Ney 1995] is to take into account that, in back-off, lower order models are used under special conditions where counts are not present in the most specific models. The relative frequencies of some words (e.g. when are part of collocations appearing in the higher order models) are overestimated in the lower order models if we do not take into account this fact.

Neural network N-grams (NNLMs)

NN LMs [Bengio *et al.* 2003; Castro *et al.* 2001a; Castro and Prat 2003; Schwenk 2007; Zamora-Martínez 2012] are easy to understand in this context since they just take profit of the general capability of artificial neural networks (ANNs) to estimate posterior probabilities. In order to estimate $p(w|h)$, a suitable representation of h is required to feed the network inputs and a way to classify among the different words of the vocabulary Ω is also required. Relating the input, it is obvious that a representation is based on the last $n-1$ lexical items makes NNLMs to be a particular case of n-grams. They have some advantages over the count based counterpart:

ANNs learn posteriors

- NNLMs are able to deal with a continuous representation of the lexicon, and the use of a factored or a structured representation is straightforward, *continuous and factored representations*
- it is also very easy to include additional inputs to represent the external context, which makes it possible the use of a cache model not limited by any a priori assumption on the use of the past history as is the case of boosting self triggered or trigger-triggered pairs; *cache based topic based*
- the inherent generalization capability of neural networks works as an automatic smoothing technique which is hopefully capable of using information of some training patterns to contribute to the estimation of similar unseen ones; *automatic smoothing*
- it is very easy to replace some inputs by categories in order to implement position-dependent class based n-grams where classes can be independently placed in the predictor or in the predicted positions of $p(w|h)$. In particular, skipping n-grams (where some elements of the past history are simply ignored) [Goodman 2001; Section 4] are just a particular case of this feature by using a special input value. This allows a novel and easy way to implement "skipping NN LMs" (see Sections 6.11 and 9.5.3); *class-based and skipping NN LMs*
- the complexity of the model (number of parameters, computational cost when training or decoding) does not grow significantly with the n-gram order, they can be more compact than some count based n-gram models. *n-gram order scalability*

but they also have several drawbacks:

- the size and the computational load grows significantly with the lexicon size, which forces the use of different tricks such as the use of lexicon subset known as "shortlist approach" or the use of hierarchical outputs; *limited lexicon size*

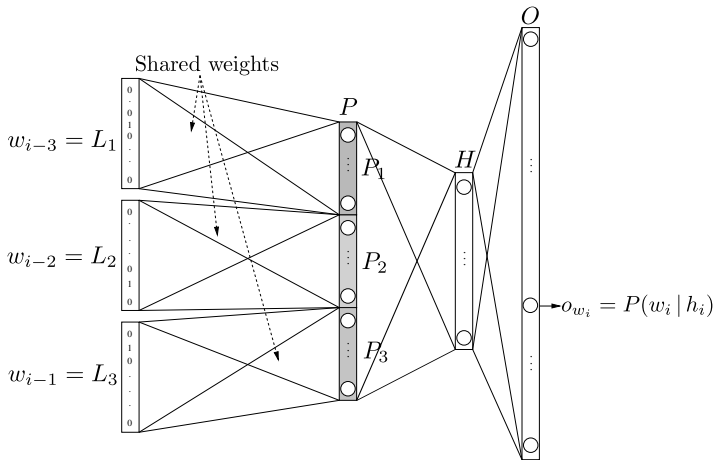


Figure 84: Architecture of the continuous space NN LM (image taken from [Zamora-Martínez 2012]) illustrating a 4-gram NN LM when computing $p(w_i|h_i)$ where the history is $h_i = w_{i-3}w_{i-2}w_{i-1}$. I, P, H and O are, respectively, the input, projection, hidden and output layer of the MLP.

- LM look-ups are more costly than count based n-grams;
- their training, usually based on stochastic gradient descent, requires a lot of computation.

Figure 84 illustrates a standard NN LM where we can observe:

factored inputs

- the input data for a n-gram NN LM are the $n-1$ previous lexical items. For each factor (just one factor in the example) a 1-out-of-n encoding is used being n the number of different variations for this factor (the size of the vocabulary, in this case);

continuous representation

- a projection layer reduces the dimensionality of the previous sparse representation and produces a continuous vector. This layer can be learnt during training so that lexical items with a similar behavior in terms of LM prediction are probably assigned nearby codes. The weights are shared for each type of projection layer (one type for each factor). During decoding, this layer can be replaced by a much more efficient look-up table. Note that there exist other approaches to learn the lexicon encoding;
- one or more hidden layers, usually with the hyperbolic tangent or with the rectified linear unit (ReLU) activation functions. Nowadays there exists a growing interest in training deep architectures, which can be easily achieved, among other techniques, by means of stacked autoencoders [Vincent *et al.* 2010];
- each output neuron estimates $p(w|h)$ for each possible w using the softmax activation function: outputs o_i are naturally constrained to lie between zero and one and to sum one, they are computed from activations a_i as follows:

$$o_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \tag{6.3}$$

It is also possible to structure the output in a hierarchical way, the shortlist approach, etc.

To summarize, non-recurrent ANNs (e.g. feedforward, MLPs,...) can be used to implement n-grams and other type of LMs with pros and cons w.r.t the count based counterpart. A great part of the literature on NN LMs is related to implementation issues concerning the estimation and use of large models. Since these are engineering problems, they are delayed to Chapter 9 (in Section 9.5) but the main ideas can be briefly enumerated:

- use of bunch mode and shuffle, linear algebra libraries to efficiently use hardware features, GPUs, etc.
- shortlist approach both at the input and at the output;
- replace the projection layer by table look-up on trained models;
- structured output [Le *et al.* 2011];
- memorization of softmax normalization constants to avoid the computation on trained models [Zamora-Martínez *et al.* 2009a];
- memorization of the partial sums of the first hidden layer;
- LM look-ahead NN LMs (novel idea proposed in this work).

6.6.2 Weighted finite automata

This section does not pretend to cover the countless aspects related to weighted finite automata (some of them were described in Chapter 5). Moreover, a review on their relationship with language modeling would be too ambitious. That is why only some details related to LMs will be pointed out, referring the more interested reader to relevant bibliography [Dupont *et al.* 2005; Vidal *et al.* 2005b]. Let us remember that n-grams are essentially a family of regular languages.[45] Indeed, several authors have proposed to use WFSTs to represent n-gram models [Riccardi *et al.* 1995; Bordel *et al.* 1997; Llorens Piñana 2000; Vidal *et al.* 2005b], an issue to be discussed in Section 9.2.

But n-grams constitute a proper subset of regular languages. For example, some WFSA can distinguish features from an arbitrarily long past history, which is not the case for a fixed order n-gram model.

Note also that, in principle, it is possible to approximate any stochastic language with arbitrary precision by means of a WFSA. That is why they constitute an attractive formalism for language modeling in spite that they are an strict subset of context free languages.[46]

Not surprisingly, several WFSA-based LMs *other than* n-grams have been proposed in the literature. Some of them are handcrafted by experts, specially for very specific semantic domains. The use of handcrafted rules is also common for constructing specific models to recognize numbers, dates, etc. which can be integrated in a larger LM afterwards. Note that, in this case, the expert usually creates the structure of the model and the weights are obtained from data in a similar way as the estimation of HMM transition probabilities.

The same occurs when the model structure is be estimated from data by means of grammatical inference techniques, which is a more

45 From this point of view, it could be admissible to place the previous section on n-grams as a subsection of this one or, perhaps, that this section should be named "Weighted finite automata *other than* n-grams".

46 Nevertheless, context free and structured LMs have been proposed in the literature. In particular, RTNs are used as LMs in [Brugnara and Federico 1997; Thomae *et al.* 2005].

general technique when enough data is available. Indeed, some grammatical inference techniques have been applied to language modeling: based on the MGGI methodology [Segarra and Hurtado 1997] or using error correction parsing [Prieto and Vidal 1992], to mention two examples. Many grammatical inference techniques are based on constructing a tree model (accepting a set of positive samples) in order to generalize them by merging similar states afterwards. Some language modeling techniques mix n-gram estimation with grammatical inference techniques. For instance, [Bonafonte and Mariño 1996] proposes to construct a WFSA from a n-gram model in order to further generalize it by applying the state merging approach.

Many grammatical inference methods are designed to estimate transducers [Vilar 2000; Casacuberta and Vidal 2004]. Their purpose is not limited to compute the probability of word sequences but to provide a translation or semantic content (see Section 6.7). Let us also remark that some concepts usually associated to n-grams can be extended to general WFSA (e.g. smoothing techniques, see [Llorens Piñana 2000]).

6.6.3 Recurrent NN LMs

Although RNN LMs and NN LMs are both connectionist LMs, they are described separatedly because NN LMs constitute a subclass of n-grams whereas RNN LMs [Mikolov *et al.* 2011] (as well as others based on different types of recurrent NNs such as LSTMs [Sundermeyer *et al.* 2012]) are able to codify the *entire* LM history in the form of the activations of the hidden layer. The fact that their internal state is based on continuous values leads to one of the main problem of RNN LM when using in decoders: the LM histories are probably never joined and the trellis associated to decoding becomes essentially a trie. The problem of dealing with long-span LM is discussed into more detail in Section 6.10.2) but, in essence, these models have been mainly used to re-rank the results obtained by a previous decoder.

Despite the differences between NN LMs and RNN LMs, they share some features:

- they can work with continuous and structured representations of the lexical items, which are automatically projected into a continuous space and, hopefully, similar items are given nearby values;
- smoothing is performed automatically due to the general capability of generalization of the models;
- they can also accept additional information coming from the external context, which makes it easier to implement dynamical LMs (e.g. cache and topic based LMs);
- they can be trained by means of gradient based techniques. For instance, with backpropagation through time;
- both can use similar techniques to speed up the computation in terms of reducing the input or the output size by grouping lexical items into classes (shortlist approach) or either by using a hierarchical output.

6.6.4 Other LM types

As stated at the beginning of this Section 6.6, this work does not pretend to be any kind complete review or taxonomy of LM types. With this in mind, the presence of a section titled "other types" is just a way to palliate this incompleteness and a way to group less relevant LM types. Similar ragbag sections can be found in other works related to language modeling (e.g. [Goodman 2001; Sect. 10]). We have left many LM types completely aside, as is the case of those based on latent semantic analysis [Bellegarda 2000] or from the result of a syntactic parser (a.k.a. structured language models [Jelinek and Chelba 2000]). Whole sentence LMs have been briefly described in section 6.1.2. There exist a highly related LM types also based on the maximum entropy approach which follows the chain rule decomposition: maximum entropy LMs [Rosenfeld 1996] are also based on a set of (usually binary) features chosen by hand although, in contrast to whole sentence LMs, they are extracted from the past history of the current lexical item to be predicted.[47]. Let us finish this section by briefly describing two type of models: On the one side, decision tree LMs are somewhat related to lattice based LMs described in Section 6.11. On the other side, we will describe a domain dependent LM in order to show that there can be quite less conventional LM types for some specific tasks.

Decision-tree based LMs

A regression decision tree[48] is a model which can be used to estimate probabilities given a training set were inputs can be converted into a set of predictor variables. Not surprisingly, these models have been proposed to estimate LMs [Bahl *et al.* 1989a] where a set of predictor variables represents the LM history h. A decision tree LM is constructed using a greedy approach based on the information gain, a concept related to entropy. This model becomes a particular case of n-gram when features associated to the predictor variables comes from the last $n-1$ lexical items. The last 20 words are used in [Bahl *et al.* 1989a]. Their main drawbacks are the use of a greedy approach and the problem of data fragmentation associated to this type of models. A more recent approach related to decision trees is the use of random forest (a set of randomized decision trees) [Xu and Jelinek 2004].

subclass of feature-based n-grams

random forest

Domain dependent LM

Some domain dependent tasks can take profit of specialized LMs which can be modeled in a more compact (or just different) way than using traditional regular or context free language formalism. We are not claiming that these languages are not part of these families, just that they could be better described otherwise. Note also that these LM types probably do not deserve the special attention we are giving them here, but we want to remark, by means of a simple example, that any list of LM types cannot be exhaustive.

47 Indeed, they can be a particular case of n-grams depending on how the past histories are clustered.

48 See `http://en.wikipedia.org/wiki/Decision_tree_learning`.

Let us show an example of a LM which is not estimated from training data and which can give an idea of what we are talking about: recognition of chess score-sheets [Baird and Thompson 1990]. As can be observed from the example of Figure 85, a simple layout analysis can be used to extract the annotation of each movement. The sequence of movements can be considered a sequence problem and each movement annotation image can be decoded using handwriting recognition techniques. Hopefully, character recognition ambiguities may be solved by considering the entire sequence of movements. The rules of chess can be used to constraint the LM used in this decoding process and techniques similar to those used by computer chess programs can be used to evaluate the quality of each movement in order to assign them a probability. Also, the representation of LM identifiers could mimic the chess board configurations. We have not found this idea in the literature, probably because the recognition of score-sheets remain a quite marginal task.

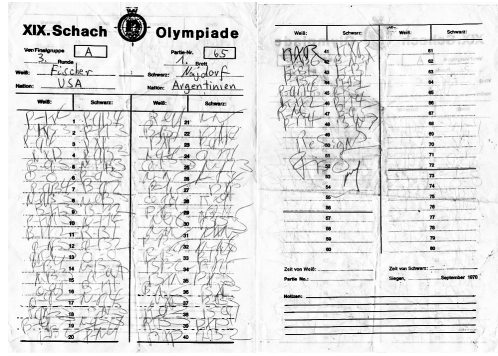


Figure 85: Example of a chess score-sheet. Recognition of this type of forms can take profit of a domain dependent LM, where the rules of chess are taken into account.

6.7 ON SEMANTICS

Let us briefly outline some relationships between language modeling and semantic interpretation. Language modeling deals with the computation of probabilities over sequences of lexical items. We saw in Section 2.5 that language understanding is task by far more complex than the joint segmentation and classification of sequences. Nevertheless, we also saw that the extraction of conceptual constituents or concept tagging is a first common step in some SLU systems where language modeling may contribute. There exists different alternatives to relate the generation of concepts C and words W:

- it is possible to recognize the word sequence and to apply a semantic model on the resulting output. The output of recognition can be represented by means of word graphs or confusion networks to deal with ambiguities. In this case, the first stage obtains the most probable word sequences and the semantic stage roughly estimates $p(C|W)$;
- we can also model the joint probability of words and concepts $p(C, W)$. From this generative model we can obtain the joint sequence of maximal probability;

- it is even possible to use a language model describing the probability of concept tag sequences $p(C)$. This model acts as the first step of a two stage generative process where a word sequence is obtained from each concept tag by means of a model $p(W|C)$. In this way, the composition acts as a language model of words and the sequence of concepts acts as the hidden labels of a HMM.

Note that the first approach does not necessarily require separate steps for decoding and for concept tagging when all models can be represented by means of transducers. But note also that we would compute the concepts associated to the most probable word sequences.

The second approach is similar to some SMT approaches. For instance, in the GIATI methodology [Casacuberta *et al.* 2005], if applied to concept tagging, word and concept pairs are converted into a sole string from an extended alphabet in order to infer a WFSA, which is very similar to the MGGI methodology [García *et al.* 1990] also used in language modeling [Segarra and Hurtado 1997]. SMT based on bilingual tuples is also similar to GIATI using n-grams.

The third approach usually assumes that word sequences generated by each concept tag are independent, mimicking the frame independence assumption of HMMs. This approach has been coined "two level" in some SLU works.[49]

Let us also remark that most LMs described in this work are based on formal language formalisms manipulating weights from a semiring. It is possible to use a tuple semiring to include values other than probabilities. Some of these semiring types can be used to manipulate semantic information in quite sophisticated ways so that some combinations can be mapped to a special *bottom* value to indicate that no valid information can be derived from them. There is nothing wrong on this approach but it should be clear that there is a certain mass of probability associated to those word sequences.

6.8 DEALING WITH OOV WORDS

Lexicon-driven unconstrained recognition systems have to deal with the problem of OOV words no matter the size of the lexicon. Words not belonging to the lexicon cannot appear at the recognizer output (and are sure errors) unless the system is able to recover them in a different way. Besides, these errors often propagate to neighboring words[50] so the error caused by them may be higher than the OOV rate. Moreover, OOV words are often content words (nouns, verbs, adjectives, and adverbs) [Bisani and Ney 2005]. Some authors have proposed methods to detect OOV words (which can be taken into account in the language model by using the 'unknown' word type as explained in Section 6.3.1.

sure errors

There are several ways to deal with the problem of OOV words in the overall decoding process,[51] but, in this section, we are interested in

49 This approach is not limited to SLU tasks but can also be used to obtain word based LMs. Paraphrastic LMs [Liu *et al.* 2012] seem a particular case of this approach.

50 By means of the language model, but also due to a bad segmentation and a bad segment (acoustic) likelihood estimation. A chain of mistakes.

51 A complete review and the quest for better solutions to deal with OOVs in the overall decoding can constitute an entire PhD subject for its own. An example of a technique

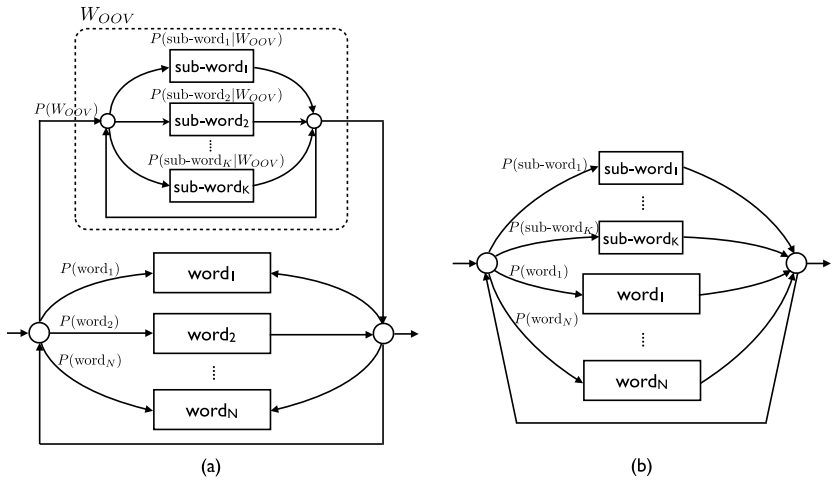


Figure 86: Figure taken from [Parada 2011; Fig. 2.5] illustrating two hybrid LMs to deal with OOVs: (a) hierarchical hybrid LMs and (b) flat hybrid LMs.

this problem from the point of view of language modeling. From this perspective, either the lexicon estimation process is able to emit word hypothesis with the <UNK> label (see Section 10.7) or the LM is able to model unknown words by means of sub-words.[52]

lexicon-free

The approach relying on sub-word based LMs [Brakensiek *et al.* 2000; Brakensiek A 2001; Wienecke *et al.* 2002; Bisani and Ney 2005; Schambach 2009] is often known as lexicon-free. Character n-grams are able to learn, to some degree, the words and sequence of words appearing in the training corpus (specially for high values of n), and also to model words not belonging to that corpus (see also Section 15.4). There exists also LMs capable of dealing with both words and sub-words, they are called hybrid LMs [Choueiter 2009]:

- hierarchical hybrid (Figure 86 (a)) LMs embeds a sub-word based LM in a word based LM replacing the <UNK> transitions. Note that the probability of entering in the sub-word based LM part must be tunes carefully;

- flat hybrid LMs (Figure 86 (b)) include both word and sub-word entries at the same level, but it cannot model word boundaries between adjacent OOV words.

We believe that it is more elegant to place the sub-word OOV detector in the lexicon estimation module instead of embedding it in a hybrid LM (as proposed in Section 10.7) since it is able to adjust the likelihood of the <UNK> hypothesis by comparing the recognized sub-word sequence with other recognized words using, for instance, Levenshtein distance.

to detect OOVs not related to LMs is the use of a recognizer without OOV detection capabilities followed by a post-process to detect OOV by means of confidence measures.

52 This solution cannot be used, or would be much more complex, when a transduction stage is associated to the LM (translation or understanding).

6.9 LM WRAPPERS

The concept of wrapper is used in Computer Science when an object delegates on another one in order to provide a different interface. The use of interfaces to access LMs is described in Chapter 9, which is devoted to language modeling from the implementation details point of view. Let us enumerate and describe some features which can be integrated in LMs in a generic way as far as the underlying LM provides the proper interface. Wrapper LMs can make transparent or, contrarily, can hidden some features. For instance, they can allow a word-based system to work with a phrase-based one. Some operations such as the LM (linear, log-linear) combination can be implemented in a straightforward way. Other features (such as expansion of categories, error correction or context dependency) can be seen as particular cases of the transducer composition approach. The following list is not exhaustive and some new applications not found elsewhere are proposed.

6.9.1 Based on transducer composition

Transducer composition, which has been presented in Chapter 5, is an elegant approach to cover all stages[53] of certain types of decoders. Since most language models are particular cases of WFSTs, it is not surprising that the LM interface described in Chapter 9 (for regular languages) is very similar to the automaton interface proposed in [Mohri *et al.* 1998]. Many LM features obtained by means of wrappers have also been described by means of transducer composition. But note that it does not necessarily imply that this is the only or the best way to implement them in any situation.

Category Expansion

A naive approach for expanding categories consists of replacing, in the category-based LM, each transition labeled with a category identifier by the set of transitions associated to all lexical items being part of the category. But note that transducer composition is a more general technique able to take into account the expansion dependent on the LM history as described in Section 6.3.3. Categories have other interesting applications such as the Out-Of-Shortlist approach to deal with the lexicon scalability problems of NN LMs.

The expansion of context-free LMs is a more complex process which enables the use of context free models as if they were regular. This is usually achieved by simulating a quite inefficient top-down parser. The basic idea of this expansion process is to associate to each LM identifier the sequence of model expansions[54] as far as the context-free grammar or the RTN has no left recursions. This technique was introduced in Section 5.5.3. Although there exist more efficient parsing techniques for parsing context-free languages, the combination of on-the-fly dynamic expansion with pruning techniques would make this approach feasible in practice.

simple but inefficient

53 From language modeling and lexicon expansion, to the generation of frames in HMMs.

54 Roughly speaking, the stack which can be implemented in a persistent way by means of a spaghetti stack with memoization.

Edition operations and error correction

Error correction analysis allows the insertion, deletion and substitution of symbols with respect to a language model in order to obtain a modified version usually more permissive. In this way, it can be considered similar to of a particular type of smoothing [Dupont and Amengual 2000]. These operations are related to the noisy-channel (also known as source-channel) paradigm [Bahl *et al.* 1983] and the error channel may be defined by specifying the probability of insertion, deletion and substitution. It is possible to specify this feature by means of weighted transducer composition using what is known as "corruption transducer" [Mohri 2003].

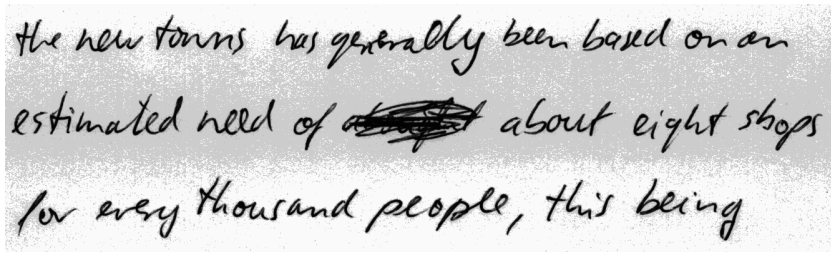


Figure 87: Example extracted from IAMdb. As can be observed, crossing outs may appear in every place with no a priori statistical relationship with word history. By combining a generic language model with a particular type of model which deals with the insertion of these units, the underlying language model can be constructed without getting worried about crossing outs.

Some examples of the practical interest of error correction LM wrappers are the treatment of silences and filled pauses in ASR and the treatment of crossing outs in HTR, as depicted in Figure 87. It is possible to use special lexical items to denote these items but it is not strictly necessary for LMs to deal with them, specially if our training corpus is not obtained from transcriptions.[55] This idea appears in some works, for example in [Finke *et al.* 1999; Section 3.2]:

> *In order to deal with filler words, i.e. words that are not modeled by a particular FSM grammar (these are typically pauses such as silence and noises), the decoder virtually adds a self loop with a given cost to each grammar state.*

Let us observe that the use of this kind of wrappers can lead to nondeterministic LMs where the set of LM states accessible from a given one can be really huge due to deletions. This is a perfect example to demonstrate the interest in including a pruning criteria in the LM interface in order to remove the less promising continuations. This will be described in Chapter 9 ans seems a novel feature that we have not found in the extensive literature.

55 Even if the corpus provides a transcription, we can simplify the task by assuming that crossing outs or filled pauses may appear anywhere independently of the content.

Context dependency

A context-dependent wrapper is a model which offers the illusion of using context-independent units and is another example or particular case of what can be easily done by means of transducer composition. It is not surprising that these features can also be emulated by means of the LM interface described in Chapter 9 since, as we will see, the proposed LM interface for the case of regular languages is quite similar as the automaton interface proposed in [Mohri *et al.* 1998].

6.9.2 Changing lexical units

It may be desirable, in some cases, to allow the use of a LM with a different lexical unit type. Some cases, such as the expansion of categories or the use of context-dependent units, have been discussed before. Let us now describe two additional examples which deserves special attention since we have not found them in the literature.

Implementing lexicon-based LMs with lexicon-free LMs

Let us imagine that we have an n-gram of sub-word lexical items (e.g. graphemes) but we need an n-gram based on words. The interest of this technique is twofold: on the one side, let us remark that NN LMs of a *very high* order (e.g. 20) can be implemented *very efficiently* (Section 9.5.3) so that it is feasible to construct with them lexicon-free LMs capable of implicitly learning even words sequences. On the other side, these models can be used to deal with OOV words due to smoothing. In this way, it could be possible to construct a flat hybrid LM only with the sub-word based LM. Moreover, this model could support the inclusion of new words *on the fly*, which is an adaptive LM technique not easy to implement in other LM types.

The basic idea is to construct the LM history of the first LM from the LM history of the second with the aid of a lexicon. For example, let us suppose that we have a character based 10-gram at our disposal and we try to emulate a word based trigram to compute $p(\text{milk}|\text{cat drinks})$. In this case, we have to: (1) obtain the LM history "t␣drinks␣" formed by the last 9 characters obtained from the text "cat␣drinks␣" and the lexicon, and (2) combine the sequence of LM lookups over the character-based 10-gram from the transcription of "milk":

$$\begin{aligned} p(\text{milk}|\text{cat drinks}) \quad = \quad & p(\text{m}|\text{t␣drinks␣}) \cdot \\ & p(\text{i}|\text{␣drinks␣m}) \cdot \\ & p(\text{l}|\text{drinks␣mi}) \cdot \\ & p(\text{k}|\text{rinks␣mil}) \cdot \\ & p(\text{␣}|\text{inks␣milk}) \end{aligned}$$

This model, as explained up until now, does not satisfy

$$\sum_{w \in \Omega} p(w|h) = 1$$

for the lexicon Ω. In order to fulfill this requirement (which is not necessary, in most cases), it would be convenient to organize the lexicon into a tree or a lexicon network, as described in Section 10.5, to save some computations. Another apparent drawback of this approach is that it seems to penalize longer words.

cross-task adaptation

In order to illustrate the flexibility of this approach, let us consider the following example that we have never observed in the literature: the use of a phone-based LM in a character-based task (e.g. to recognize a handwritten text by means of a phone-based LM). The lexicon estimation process would use the grapheme-based lexicon representation and the emulated word-based LM would rely on the phonetic transcription. There exists other similar ideas to use phonetically motivated techniques for HTR which could be more robust against some spelling, word separation or grammatical mistakes.

Implementing lexicon-free LMs with lexicon-based LMs

This is the opposite of the problem described before: using a lexicon based LM as if it was a lexicon-free LM. Although it may not seem very practical, at a first glance, at least two different applications can be easily envisaged: (1) as an alternative way to decode from a DAG of sub-word units. This approach is alternative to first converting the sub-word sequences into words, as described in Section 11.2, and to apply the original LM afterwards. And (2) to mix with a real lexicon-free LM to improve the performance of lexicon free tasks. Let us just remember that a properly tuned linear combination of LMs cannot hurt.

A lexicon is required to construct a lexicon-free over a lexicon-based LM. This lexicon can be efficiently represented with a tree or with a lexicon network, as described in Section 10.5. The LM history of the wrapper LM is composed by the LM history at the word level and the prefix of the word (a state of the lexicon network). Note that a character sequence can be the prefix of several words and, in this case, is is important to marginalize the probabilities of all those words, which would correspond to the use of the LM look-ahead technique described in Section 6.10.1. Note also that character sequences which do not correspond to any word prefix would receive a zero probability unless some smoothing technique is applied. Note also that this LM is non-deterministic, in principle, unless determinization techniques were applied over the above description.

6.9.3 Partial/imperfect transcription

In some recognition tasks, as is the case of the transcription of broadcast news, the speaker usually follows a script. However, she/he does not necessarily follow the transcription verbatim. We talk in those cases of partial or imperfect transcriptions. It is desirable to take into account this information during decoding although this problem is not the same as alignment or segmentation. Some works have studied this issue. For instance, [Lecouteux *et al.* 2008] proposes to modify the decoding scheme of an A* stack decoder to this end. We believe that there can be decoder-agnostic solutions by tackling the problem at the LM level. The novel proposed technique is not equivalent to that of

decoder-agnostic solution

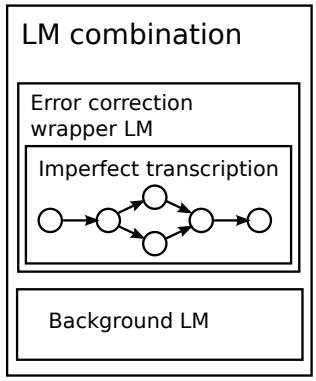


Figure 88: LM for imperfect transcription decoding: a finite state automaton is obtained from transcription alternatives. An error correction wrapper makes it possible to deal with edition operations. This model is combined with a general background LM. The overall LM is nondeterministic because of deletions (a determinization process would fix it).

multiple transcriptions

[Lecouteux *et al.* 2008] and makes straightforward to take into account multiple transcription alternatives. It consists in combining the following components (as depicted in Figure 88):

- a general language model, just the same we would have used without this technique;
- a LM constructed by placing an error correcting wrapper LM over the imperfect transcription. Note that this transcription can be considered a particular case of LM which can be represented by means of a finite state (aclyclic) automaton which would only accept the set of transcription alternatives.

The LM identifier would comprise two values: the LM identifier of what has been transcribed so far and a position in the automata of alternative transcriptions. This technique can also be used during training when a perfect ground-truth is not available.

Note that information sources other than a detailed transcription may guide the LM: A rough summary consisting in an ordered list of topics could be used to drive a topic mixture LM where weights traverses this list. Similar ideas can also be applied to interactive transcription tasks.

interactive transcription

6.10 SOME LM DECODING FEATURES

LM look-ups are costly. They are mainly performed either by LM toolkits to measure the test set perplexity, during re-ranking or during decoding. The two latter cases are more critical since the number of LM evaluations can be quite high.[56] In some systems, the LM module is responsible of providing the set of lexical items to be matched with the input. In a different, more usual setting, the module responsible for lexicon estimation (see Chapter 10) generates a collection of reasonable words that may follow the current hypothesis and the LM look-ups are restricted to this set afterwards. Finally, in the transducer composition paradigm, both information sources are simultaneously taken into account in order to precompute an static network. From this brief description, we can only catch a glimpse of the possibilities

56 It depends on the number of LM histories associated to the set of active hypothesis.

of using LMs in decoding. It will become clear, in following chapters, that LM implementations can provide operations other than just computing the probability of a lexical item given its past history.

LM implementation details are mostly delayed to Chapter 9 since this belongs to the second part of this writing.[57] However, some features related to decoding may be relevant in this part of the work.[58] The reason is that we would like to suggest, in Section 6.11, the possibility of a system capable of reasoning about the features a LM can display (and how they can be better provided) given the features of its constituents and how they are combined. The consideration of a *type system*[59] of LM sub-modules[60] is a modest proposal since it is not intended to obtain LMs with better perplexity but rather to be able to automatically optimize the representation and evaluation of complex LM systems made from several parts. This reasoning would also take into account which features are required by the decoder or, conversely, used to determine which decoder is best suited or at least to tailor it.

6.10.1 Pruning, bounds, sub-lexicons, look-ahead and scales

One of the features that a LM system can specify, other than just computing the probability of a lexical item following its past history, makes sense in the context of decoding where the worst hypothesis can be pruned.[61] In this setting, LM probabilities lying below a given threshold can be discarded without actually computing them. The capability of taking into account this information inside the LM system can help to reduce the overall computational cost in many cases.

Another useful way to help decoding is to obtain the set of words whose probability is higher than a given threshold:

$$\Omega_{higher}(h, \rho) = \{w \in \Omega \mid p(w|h) > \rho\}$$

lattice of tree-lexicon decoders

It is possible to predetermine a finite collection of these lexicon subsets, organize them into a lattice algebraic structure[62] so that given a set of hypotheses, we can: (1) obtain the most specific useful lexicon subset (from the LM point of view) for each of them, (2) determine the least specific one in a very efficient way from the lattice, and finally (3) use this particular lexicon in the lexicon estimation process. This idea is further developed in Section 11.1.3, but the LM has to provide this feature to be able to use it.

Another useful value that can be computed from a LM history is the following upper bound:

$$\mathrm{LM}_{max}(h) = \max_{w \in \Omega} p(w|h)$$

57 It has been divided into three parts trying to mimic the successful trilogy, see Section 1.2.
58 Other relationships between language modeling and decoding, as is the case of the grammar scale factor, are described elsewhere.
59 A set of types and rules to reason about their properties, as is usually understood in the field of functional programming languages.
60 Not only LMs, their constituents can be feature extraction modules, etc. as better described in Section 6.11.
61 Pruning is described into more detail, for instance, in Section 10.2.5.
62 A partially ordered set.

which can be used by the decoder to prune hypothesis without using lexicon estimation or, alternatively, can be used to better prune during the lexicon decoding process. It is also possible to restrict this upper bound for a given lexicon subset Ω':

$$\mathrm{LM}_{max}(\Omega', h) = \max_{w \in \Omega'} p(w|h)$$

and to compute the corresponding lower bound as well:

$$\mathrm{LM}_{min}(\Omega', h) = \min_{w \in \Omega'} p(w|h)$$

The first can be used to implement the LM look-ahead described below and also in Section 10.6.7. The combination of both upper and lower bound can be used to implement submodel dominance recombination, described in Section 10.6.8.

LM lookahead

LM look-ahead (LMLA) [Ortmanns *et al.* 1996] is a useful feature to obtain an estimate of the LM transition probability $p(w|h)$ when the word is not yet known but a set of possible alternatives (the more reduced the set, the better) can be established. LM look-ahead can be used during decoding to improve the pruning of the search space, as described into more detail in Section 10.6.7. It is based on a more general computation: marginalizing the probability of a subset of words. The case of LMLA usually refers to the marginalization of probabilities of words starting with a given prefix and is usually employed in prefix-tree lexicon decoders. But other uses of the marginalization of LM probabilities may be useful: for instance, to compute $p(<oos>|h)$ which may be useful in one of the procedures to implement the shortlist approach (see Section 6.3.3). In most cases, many of these summations can be computed beforehand since we can know in advance the set of lexical items we have to add and we can efficiently store these values associated to states of an automaton, etc. LMLA may also benefit from the fact that LMs are based on back-off smoothing to represent LMLA information in a sparse way [Nolden *et al.* 2011a]. The use of a LM wrapper to implement a word-based LM by means of a subword-based LM makes the implementation of LMLA immediate. Related with the LMLA, a novel type of NN LM is proposed in Section 9.5.4. This new model can directly compute the LM look-ahead by taking profit of the capability of a MLP to have several independent softmax outputs, each one associated to a different branch of a tree lexicon.

history dependent LM scale

Another LM feature specially suited for LMs with a costly normalization factor (as is the case of those based on a softmax output) is the capability of computing scaled probabilities, meaning that these results could only be compared among different LM evaluations associated to the same LM history. These values could be used in some particular cases when working with a confusion network produced by another system (with another LM). In a more general case, the system could compute the minimal set of scales required to disambiguate among hypotheses associated to different LM histories. This feature seems to be not reported elsewhere in the literature.

6.10.2 Managing LM histories and dealing with long-span LMs

On managing LM histories

In many LM papers the main motivation for the "clustering of histories" approach is to reduce the number of parameters to improve estimation. Another important reason is to reduce the number of hypothesis considered during decoding. A LM where different prefixes cannot lead to the same LM history is not as efficient as, for instance, a bigram where all hypothesis associated to the same last word are summed (e.g. maximized) in the semiring framework or joined in dynamic programming.

It is convenient to distinguish between the LM history strictly required *to compute the features for estimating the probabilities* (e.g. the input to the MLP in a NN LM) and information required *to update the LM history with a new lexical item*, unavoidable during decoding. This distinction is not important when our main aim is to compute the test set perplexity because we have unambiguous sentences in advance and the complete prefix is there for free. The distinction is not required either when using n-grams, the most predominant LM type, because both of them coincide. But let us see some LM types where this difference is evident:

poor man's higher order n-gram

- skipping-n-grams [Goodman 2001; Section 4] are an extension of n-grams which allow to skip certain items of the past history. They are usually combined and, for instance, a mixture of skipping n-grams may require much less parameters and can be faster to compute than the equivalent n-gram;
- bag-of-words n-grams is a quite simple idea: just take into account the last $n-1$ lexical items but ignoring the order.

Skipping-n-grams can be seen as a particular case of n-gram where some lexical items from clustered history are ignored when estimating the probability of the next word. For example, when computing the probability of the following sentence with a 4-gram:

<BOS> THE BROWN CAT DRINKS SOME WHITE MILK <EOS>

the LM estimates, at a given moment, $p(\text{DRINKS}|\text{THE BROWN CAT})$. An skipping 4-gram ignoring the second word in the clustered history would compute, instead, $p(\text{DRINKS}|\text{THE \_\_ CAT})$. An advantage of this approach is that it may generalize in ways different from backing-off to a lower order n-gram, as illustrated by the following training fragment which could contribute to the estimation of the above probability:

... THE WHITE CAT DRINKS WATER ...

Despite only requiring THE __ CAT to estimate the probabilities, we cannot drop BROWN from the history since *we need it to update the LM history* with DRINKS to obtain BROWN CAT DRINKS. This example illustrate the idea that the minimal information required to compute the LM probabilities is not the same as the minimal information required to update the LM identifier with a new lexical unit.

Dealing with long-span LMs

The terms "long span" or "large spans" refer to models where the LM history cannot be obtained from the very last lexical items. An example are RNN LMs, whereas LMs excluded from this category are lower order n-grams.

The use of long span LMs to obtain the test set perplexity of a set of sentences and to re-rank n-best lists is straightforward. However, long span LMs pose problems in re-ranking of word graphs and in decoding since, as it has been pointed out before, hypothesis can only be merged when they have associated the same LM identifier. In the case of using RNN LMs, it is highly unlikely that different word prefixes lead to the same RNN continuous internal state. In this case, the structure of the dynamic programming trellis search graph would degenerate into a tree which rapidly becomes too huge excepting for the use of pruning techniques. Indeed, long-span LMs is one of the subjects of [Deoras 2011], where two solutions are proposed:

- iterative decoding: a type of hill climbing when a local search is performed over a word graph, at each iteration, in order to detect regions of low confidence called "islands of confusability";
- to replace the long-span LMs with a similar LM of controlled complexity[63] (using the Kullback Leibler divergence to measure similarity). This LM is used in a first stage to obtain a reduced set of solutions (e.g. an N-best list[64]) which is re-ranked with the original long-span-LM. The rationale for using an approximated LM is to reduce the mismatch between both stages.

Most other solutions found in the literature are based on *truncating* the past history when merging hypothesis. For instance, long-span LMs are used in [Liu *et al.* 2014; Sect. 3.1] to determine the probabilities based on the full history, but a LM cache is used and probabilities are associated to the truncated history and hypothesis are merged based on this truncated history as well. This is a sub-optimal approximation since a different full history is probably used to evaluate the LM contribution. The rationale behind this approximation is that different LM context sharing the same truncated history are probably similar.

A decoding technique described in Section 8.4.4 allows the use of a fist decoder based on a cheaper lower order LM to generate a pruned word graph where LM scores has been removed which is feed to a second decoder[65] while the input is being processed without requiring to finish the first stage in order to start the second one.

Other technique to deal with RNNs, described in [Liu *et al.* 2014; Sect. 3.2], is based on clustering similar representations of the RNN internal continuous state. Other works also based on clustering the internal state of a RNN propose the transformation of the RNN into an automaton beforehand.

63 Note that the general idea of training a LM on the basis of a second LM is not at all new: this idea has also been described in [Nederhof 2005] to train a finite automaton by using the estimations of a PCFG and vice-versa.

64 It seems dummy to mention that it is straightforward to organize the N-best list in a trie to share common prefixes.

65 In this case, the term "reranking" is usually employed.

6.11 HETEROGENEOUS LATTICE-BASED LM

Nearly at the end of this chapter, after a careful exposition of ideas, we hope that the following proposal seem as trivial[66] as possible:

- we have shown that many LM types are particular combinations of features relating the use of lexical units, categories, type of probabilistic decomposition, combination of different LMs, inclusion of adaptation information, etc.
- we have distinguished between count-based and feature-based n-gram models and we argue that feature-based are more appropriate to deal with dynamic models because the external context can be included as additional features;[67]
- there is a mismatch between: (1) the LM information required to compute the above features, and (2) the minimal information required to determine which hypothesis can be joined during the search (e.g. dynamic programming). This information is updated when a new lexical item is hypothesized and we have shown examples (e.g. skipping n-grams, bag-of-word n-grams) where different hypotheses can have the same set of features to compute LM probabilities and, hence, the same LM scores;
- the same problem of LM histories mismatch appears when combining different LMs (e.g. when combining n-grams of different orders), and it is also possible to reason about the relationship between different LM histories (e.g. the history of a 2-gram may be inferred from the history of a 3-gram);
- we have observed that language models can show other properties besides the computation of the probability of observation of a lexical item given the past history. For instance, some LM have special capabilities to compute the LM look-ahead, to apply LM pruning or to select of a sub-lexicon tailored for a given set of LM states, etc.

ground basis vs tricks

Although these topics seem more related to implementation details and will be discussed from this point of view in Chapter 9, it is interesting to formalize these LM properties as well as the relationship between different kinds of LM histories and features. We believe that these relationships *corresponds to a partially ordered set or lattice structure*.

The idea of organizing LM histories in a lattice algebraic structure is not new: it has been proposed in [Dupont and Rosenfeld 1997]. This was later generalized in [Bilmes and Kirchhoff 2003] dealing to a "back-off graph". In both cases, lattices are restricted to a hierarchy of clustered LM histories: In the case of the back-off graphs, the concept of descending to a lower order model is generalized to that of discarding different factors at different positions of the past history. We believe that the idea of a lattice can be further generalized by extend-

66 When something does not seem trivial, it is usually because of a wrong presentation of preliminaries. This problem, when found in some articles, may be due to a lack of space. In other cases is due to a mismatch between authors and readers' background, bad pedagogical capabilities of writers (or, remembering Nietzsche's quote *"They muddy the water, to make it seem deep"*) or readers limitations, every combination is possible.

67 The features summarizing the external information (prior user turns in a dialogue system, etc) is usually known as the *cache*.

ing the lattice so that it not only contains different LM histories but also feature extraction modules for a feature-based LM. The current proposal is restricted to the chain rule decomposition and clustering of histories approach,[68] but it is not limited to n-grams since some history managers can also be finite state models other than n-grams and even their continuous counterparts (e.g. RNNs). Some examples of feature extraction types are:

- bags: which receive an ordered set of features and remove the order, when used with the last $n-1$ lexical items of a given history, this would lead to a bag-of-words n-gram model;
- position dependent categorization, which means that each factor of each lexical item is categorized depending on their position relating the item to be predicted. Since the identity function and the constant function can be seen as two particular cases of categorization, skipping-ngrams are a particular case of feature-based LMs using this approach. Skipping NN LMs are described in Section 9.5.3;
- topics: some topic tracking feature extractors produce an estimation of the probability of being in a particular topic at a given moment. These estimations can be used to construct a topic-mixture LM of directly as input of a feature based classifier,
- a cache summarizing recent past turns, this feature is not modified during the decoding of the current sentence. This idea could be extended to deal with information from the current sentence as explained in Section 6.5.

Most features can be constructed from other ones and, ultimately, from LM histories and external contexts. These lattice-structured types of features can be used as elements to construct feature-based LMs in a similar way as the use of control flow graphs to describe neural networks. This flow graph would have the form of a DAG[69] where one of the nodes is the final LM and others are used as auxiliary components.

Interestingly, since LMs can be viewed as function from features to scores which, in turn, can be considered as new features by themselves, it is possible to include LMs in these DAGs in a nested way leading to a homogeneous representation: feature-based LMs can also be seen as parts of the hierarchy.

Although these ideas are just a sketch (see also Section 9.1.4), our conviction is that a formalization of the different components and their relationships would enable an automatic reasoning in a similar way as many programming languages allow sophisticated type inference. For instance, it should be possible:

68 The extension of these ideas to whole sentence LMs and to the "combining spans" approach (context free models) is not yet refined.

69 A DAG because it does not seem reasonable to allow circular dependencies. Similar ideas are proposed in the literature to implement gradient based neural models by means of "flow graphs". The same approach is proposed in this work to describe recognition engines by means of a dataflow architecture (Chapter 8) or to describe segment models, not to mention the specification of the problems by means of reifying annotation graphs described in Section 3.3. The final goal is to join all these ideas at the end of this writing (Conclusions in Chapter 16) to try to put in practice some thoughts from Section 1.3.

- to determine the minimal or least common history manager for a given set of features;
- to simplify, if possible, a complex LM by removing redundancies,
- to infer if a given LM is coarser than another one or to detect if several LM histories can made use of the same evaluation of features in a predictable way;
- to determine which features (LMLA, etc.) can be efficiently deployed.

Organizing LM history extractors and feature evaluations into fine grained and more coarse models can be used to automatically group fine grained ones into coarser clusters to deal with long-span LMs as described in previous section. Another application of this grouping of LM histories is the use of an alternative way to apply look-ahead (more akin to the fast-match look-ahead described in Section 10.6.2 than to the LMLA described in Sections 6.10.1 and 10.6.7) in language modeling: the decoder would proceed during a determined *time*[70] and only those hypotheses which have survived are used to control which long-span LM histories are promising.

Ultimately, it should be possible to automatically determine whether it is possible or not to offer the transducer composition interface and to determine which parts can be simplified at "compile time" and even to separate the parts can be tacked by the static transducer composition from the others that should remain dynamical.[71] For instance, models based on features depending on an external context are better suited for a dynamical treatment.

6.12 SUMMARY AND SOME CONCLUSIONS

This chapter has introduced some language model concepts assuming a previous familiarity with weighted formal language formalisms previously described in Chapter 5. We have centered our attention on the estimation of the a-priori probability of a sequence. Other LM approaches, such as possibilistic LMs [Oger and Linarès 2014] or some kind of discriminative[72] LMs have not been taken into account.

not a review

Obviously, this chapter is not intended to be a review of language modeling since we have left out many important aspects as LM estimation or adaptation. Moreover, the focus has been on the use of these models for decoding, disregarding other practical utilities as their use in information retrieval tasks.[73] Implementation details have been post-

70 The use of word "time" is misleading when talking of sentences in general, not limited to time series. Moreover, the parsing process has been described as using an input DAG instead of a sequence. We abuse from the "time as position" analogy and use sometimes "moment" to refer to a vertex of the input DAG.

71 The question "dynamic vs static decoders" can only be posed where static decoders can *also* be applied.

72 Discriminative LMs [Okanohara and Tsujii 2007] are intended to classify sentences as correct or incorrect assigning them a score which is not necessarily related to probabilities and can be used to rank a set of alternatives. Not to confuse with the highly related discriminative training of probabilistic LMs [Kuo *et al.* 2002].

73 The interested reader can find a large number of set of the *same old, same old* descriptions of language modeling, many of them mainly focused on count based n-grams and their uncountable types of discounting techniques.

poned to Chapter 9 in order to place them in the proper part of this writing.[74] These more engineering aspects will cover LM interfaces, efficient representations, etc.

We have tried to organize a large set of particular models, many with coined nomenclature, as a combination of orthogonal features. Special attention has been paid to separate the different lexical unit types from their actual usages in LM types. We have also tried to describe LM combination and dynamic capabilities as orthogonal features and not as LM types. Although quite unlikely, this perspective could make the process of discovering new combinations easier, which is one of the aims of this work, as explained in Section 1.3.

dynamic LMs benefit dynamic decoders

Dynamic LMs may take into account adaptation data not available during training which may even include the current sentence. The importance of dynamical features has been emphasized and techniques are described in general (not attached to any particular type of LM, although it is clear that they are more easily integrated in feature-based n-grams than in the more traditional count-based counterpart). We do not hide our interest in promoting dynamical features because they make less interesting the static transducer composition approach and favor dynamic decoders which are the ones chosen in this work.

After describing several LM types, n-grams have been divided into "count-based" and "feature-based", being NN LMs the most prominent example of feature-based n-grams.

LM & decoding relationship

We have also payed special attention to the relationship between language modeling and decoding, which is sometimes overlooked in the literature. We have detailed how some features usually performed by modifying decoders can be tackled by means of language modeling and we have put two examples (the case of partial/imperfect transcription and the interactive transcription paradigm) proposing decoder-agnostic solutions.

Some LM examples have allowed us to distinguish between the LM history required to extract features and the information required to update the LM hypothesis during decoding. This distinction leads to the proposal of a view which unifies some previous models in the form of a feature-based model using components organized in a lattice. Lattice-based LMs are not new, but we believe it is better to consider also lattices comprising not only LM histories but also features and even LMs themselves. We believe that this approach has some resemblances with the use of signal flow graphs to specify gradient based neural networks. In particular, it would allow to automatically reason about the best configuration to mix certain LMs. Due to obvious restrictions on the capabilities to develop all the ideas described in this PhD. report, our goal is to go deeper into these ideas in a future work in order to extend the notions of transducer composition to deal with the internals of these structures. We believe that it is possible to consider a mix between dynamical and static decoders taking the best of each world. In this way, future systems will be able to automatically infer which parts of the system can be processed and optimized beforehand and which others should remain dynamic.

delayed to future work

74 As stated before, each part corresponds to a different point of the "successful trilogy": 1) theory/models, 2) engineering/algorithms, and 3) tasks/know-how/experimentation.

7 SEGMENT MODELS

¿Qué tramas?[1]

CONTENTS

7.1 INTRODUCTION

This chapter reviews the most common models used to estimate the likelihood of observing a segment (in the form of a frame sequence) assuming that is was produced by a given model. These segment models (SM) correspond to the second step of the generative model described in Chapter 4 which basically approaches the joint segmentation and classification of sequences using the *beads-on-a-string* metaphor. The study of these models in this work has been focused on recognition, leaving aside generative tasks such as text-to-speech (TTS) which may also make use of them [Zen *et al.* 2009]. The problem of model estimation has been skipped as well, whereas decoding will be delayed to next chapters, specially to Chapter 10.

1 Joke playing with the ambiguity of Spanish word "trama" which means "frame" but also the verb "tramar" meaning "to plot". In this way, "Qué tramas?" means "Which frames?", but also "What are you up to?".

Some previous remarks

Before trying to organize the types of segment based model types, let us first start by pointing out some remarks. Some of them are probably clear after Chapters 4 and 6:

HMMs remain mainstream

- the success of HMMs, introduced into speech recognition during the seventies [Baker 1975; Jelinek 1976] and impulsed by [Lee 1988], has virtually eclipsed the rest of alternative segment modeling techniques for decades. Not surprisingly, most reviews on acoustic models concentrate exclusively on HMMs (enumerating a large number of model variations and extensions) and basically ignore other SM types. Other reviews mention SMs in a final section titled *"Other approaches"*.[2] Unfortunately, a coherent view of SMs places HMMs as a particular case of both SMs and DGMs. Besides, these works usually mix generative and discriminative SMs without clearly distinguishing them.
  One of the reasons why SMs *other than HMMs* have not become mainstream, not to mention to replace HMMs, is that HMMs work well in practice for may recognition tasks and the empirically measured improvements obtained with other SMs do not compensate for the large increase in the number of model parameters and in the computational complexity of the associated estimation and decoding algorithms.[3] On the other hand, it is widely acknowledged that recognition results obtained with HMMs are still far from what can be achieved by human listeners, at least in noisy conditions. Although this can also be due to semantics (LMs), there is room for improvement in acoustic modeling and this has often encouraged the proposal of alternative SMs;

room for improvement

- segment models, as used in this work, assume a particular way to solve the joint segmentation and classification problem on sequences. But this is not the only way as stated in Section 4.4 where some alternative techniques are discussed;

overloaded notation

- the term "segment model" covers quite different models. Some of them are generative and others are discriminative. Some are associated to frame sequences and others to a fixed length feature vector associated to each segment *no matter its length*. The last case has been coined as "graph based observations" by some authors [Glass 2003] although the alternative expression "segment-based observations" seems, in our humble opinion, more appropriate. The difference between this representation and the more common frame-based one has been addressed in Sections 2.9.1 and 4.4.2. We saw that approaches based on segment-based observations are usually discriminative and have some fundamental flaws since scores from different segmentation paths are not comparable, in spite of lots of simplifying assumptions, because

fundamental flaws

2 Likewise, DGMs are often placed in a different section, hence misleading the inattentive reader who might believe that this is a mutually exclusive classification.

3 Both in terms of difficulty of implementation and computational resources (time, space) to execute them. This remembers us how difficult is also to overcome statistical n-grams in language modeling. Observe also that HMMs benefit from a large experience converted into a valuable *know-how* (tuning of parameters,...) so that new approaches have probably to traverse a temporary increase of recognition rates [Bourlard *et al.* 1996].

they come from different observation spaces. Note also that this problem only happens when there may exists segmentation ambiguities, which is the reason why variable rate frame preprocessing techniques do not pose problems on this point. The problems of segment-based observations are the reason why we have focused our attention on segment models emitting frame sequences,[4] HMMs being the most notable particular case;

- nevertheless, we will see in this chapter a proposal overcoming the limitations of discriminative segment models, in general, and those based on fixed-length representations, in particular. The idea consists in combining them with a frame-based generative model instead of using them isolatedly. A particular and widely known case which deserves special treatment is the use of discriminative models *at the frame level* which can be used as frame emission estimators in what are usually known as *hybrid* HMMs. These frame posteriors are converted into scaled likelihoods by means of Bayes theorem, as described in section 7.5.3;
- the distinction between generative and discriminative models has not to be confused with the use of discriminative techniques to train or to estimate generative models. Also, some features which do not fit with a strict generative point of view can be used in generative models obtaining empirical improvements as is the case of the inclusion of dynamic features such as delta features or, more generally, the inclusion of information from neighboring frames when estimating the frame emission likelihoods. The use of this context is common practice in hybrid HMMs.

Chapter structure

The structure of this chapter is as follows:

- the first section reviews the main limitations of HMMs which have historically motivated many SM types;
- the weak duration modeling capability of HMMs leads to the division of SMs between explicit and implicit duration modelling techniques. This classification is addressed in Section 7.3 which is also related to HMM topologies, briefly reviewed;
- the following section is devoted to frame sequence emissions.
- Since the emission of just one frame is a notable particular case (and a basic piece used in the former section), it deserves special treatment in Section 7.5;
- Section 7.6 gives a rough outlines of context dependency;
- the use of discriminative models in a generative framework is the subject of Section 7.7;
- a general scheme of SMs is succinctly presented in Section 7.8. It tries to provide a kind of generative description where most SMs found in the literature can be placed as well as a method to obtain new ones;
- a final section summarizes ideas and draws some conclusions.

4 Let us remark that, ironically, most fixed-dimensional segment-based observations are obtained from a previous frame-based representation.

7.2 OVERCOMING THE LIMITATIONS OF HMMS

HMM $\subsetneq$ SM

Generative segment models are traditionally described as *opposed to* HMMs. One of our goals is to clarify that HMMs are rather *a particular case of* SMs. Probably, the main reason to historically present segment models as opposed to HMMs is to emphasize the fact that they try to overcome the main limitations of HMMs.[5]

Let us remark some features of HMMs and to expose their limitations in order to better understand how segment models have historically tried to overcome them:

HMM $\subsetneq$ DBN

- HMMs are DBNs, but not all DBNs are two-stage generative models (see Section 4.4.1). Conversely, some models proposed in this chapter, as the combination of a generative segment model with a discriminative one, cannot be described as DBNs in a simple way. DBNs are based on unrolling or unfolding templates and the use of "switching parents" allows the description of state switching processes, but this seems naturally restricted to regular languages. This also explains why, despite of the fact that most SMs can be expressed as DBNs, DBNs are not proposed as a general alternative completely subsuming SMs;

interleaved vs stacked

- HMMs admit two alternative descriptions *when generating finite length[6] sequences:* (1) a loop where state transitions and observed emissions processes are interleaved, and (2) two stacked processes, the first one generating a sequence of hidden labels and the second one the observed sequence;

- the stacked description of HMMs means that they can be considered just "miniature two stage generative models" where: (1) the first stage corresponds to a model generating sequences of "type of frame emission" or probability density functions (pdf's). This model is based on the Markov property (described, in the context of language modeling, in Section 6.1.1) and is best represented by means of weighted finite state models, and (2) frames are emitted by each type of emission pdf as independent outcomes, which severely limits the capability of HMMs for modeling the frame correlation observed in many real tasks.

  Note that there may exist a correlation between frames even if they are emitted by independent outcomes of partially correlated pdf's. Indeed, HMMs can, in principle, approximate any other model to an arbitrary precision [Bilmes 2006], but this would be at the expense of a great model complexity, which seems impractical from both estimation and decoding points of view;

- observe also that other DBNs do not admit the stacked approach when, for instance, hidden labels depend also on observed signals, as is the case of MEMMs [McCallum *et al.* 2000].

5 While staying in the *beads-on-a-string* paradigm: They have just made the beads larger. Nevertheless, some models try to explicitly represent continuity constraints across segments, as is the case of super-segmental models [Deng *et al.* 2006].

6 Remember the remark from Section 4.2 relating infinite length sequences. We can also draw attention to the compatibility of both approaches described here are compatible with ancestral sampling [Bishop 2006].

The main limitations of HMM that have motivated the proposal and development of segment models can be summarized as follows:

- piecewise stationary;
- frame independence assumption;
- Markov modeling of sequences of types of frame emissions;
- weak duration modeling (e.g. self-loop states are limited to geometric duration distributions).

Let us deep into each one in the following subsections.

7.2.1 Piecewise stationary

The notion of stationarity referred to stochastic processes means that certain statistical properties are shift-invariant. Strictly speaking, a finite length segment is not stationary even if it is a truncated portion of a stationary process. Signals produced by mechanisms that change over time, as is the case of Speech, are not stationary. The assumption of piecewise stationary means that the observed signal is made[7] from portions taken from a set of stationary and independent random processes. This is in clear contrast, in the case of ASR, with the smooth and continuous variations of the speech articulators.

The piecewise stationary assumption in HMMs refers to the fact that there are instantaneous transitions between states associated to different emission probability functions.[8]

Observe that this assumption, in speech, is also related to the feature extraction process: the most common feature extraction methods try to estimate the shape of the spectral envelope of the speech signal relying on a time-frequency decomposition to extract the instantaneous information of the frequency. But this process is inherently limited in resolution by the Heisenberg–Gabor limit. In practice, most ASR feature extraction techniques divide the speech signal into overlapped frames and rely on the Short Term Fourier Transform (STFT) to analyze them. This process assumes that the signal is piecewise stationary within each frame. Even if the speech signal is not stationary, the variation within the frame is smooth enough to make it an accurate approximation.

We have seen in Section 2.8 that speech signals are continuous signals produced by a physical source-filter model which can be described by means of a limited set of continuous parameters relating, among other things, the position of articulators. The observed speech signal can be considered the surface form of an intermediate sequence in a low dimensional *continuous* articulatory space. Some models, as is the case of the Hidden Dynamic Models (HDMs), assume an intermediary level of dynamics of the speech production and explicitly takes it into account [Deng *et al.* 2006].

surface form of hidden dynamics

Despite studying the joint segmentation and classification problem on sequences in a *task independent way*, we have to remember that most works on SMs other than HMMs have historically been limited to ASR,

7 Formally, by means of a switching parent choosing the different sources.
8 Remember that these pdf's may be tied among different states.

with some exceptions on HTR (e.g. [Artieres *et al.* 2007]) and bioinformatics (e.g. [Schmidler *et al.* 2004]). In this way, we can distinguish between tasks with discrete observation symbols (as is the case of bioinformatics) and others associated to continuous signals.

Although some SMs may make use techniques to describe continuous trajectories, there is an inherent limitation in the basic TSGM since we can tackle the piecewise-stationary and the frame independence assumptions *within a segment* but not when changing from one segment to the next one. Models extending SMs to deal with continuity constraints across segments have been proposed in the literature [Deng *et al.* 2006]. Although the more complex models described in the hierarchy proposed in Section 4.2.1 could, to a certain extent, model these constraints, the use of these extensions has been finally excluded from this work.

Nevertheless, problems such as the smooth transitions between diphthongs may be partially tackled by creating SMs representing groups of consecutive labels. These problems can also be alleviated by means of context dependent units. Observe that the use of dynamic features taking the information from neighboring frames into account violates the frame independence assumption and may operate across segments.

7.2.2 Frame independence assumption

The frame independence assumption of HMMs means that each observed frame, even neighboring ones, are the result of independent *throws of dice*. Note that this is compatible with the capability of perfectly modeling the statistical dependence among features within the same frame.[9]

This assumption can be compensated by the use of discriminative training techniques [Wegmann and Gillick 2010; Gillick *et al.* 2012] and the inclusion of dynamic features. HMMs with augmented features work well in practice but are inconsistent. Relating this fact, it is convenient to remember an alternative point of view remarked in [Bridle 2004] which consists in viewing HMMs with dynamic features as an *unnormalized generative model* where samples are constrained to be consistent relating the augmented features. Also, when dynamic features can be described by means of a deterministic mapping from the static ones, as is the case of first and second derivatives of cepstral coefficients in ASR, it is possible to employ the dynamic features while considering that the static part is the original set of observation. This can be formulated as a trajectory model [Zen *et al.* 2007].

deterministic mapping

Most proposals which claim to take the frame correlation explicitly into account, in HMMs, basically conditioned the emission of each frame not only on its associated state but also on the previous frame [Wellekens 1987; Paliwal 1993; Kim and Un 1997; Qing *et al.* 1999]. This has been later generalized by considering HMMs as a particular case of DBNs with additional edges between nodes representing consecutive observed frames, as is the case of buried HMMs [Bilmes 2003].

9 Some works using HMMs for ASR go further by assuming that individual features within a frame are conditionally independent (due to the decorrelation techniques of some feature extraction modules) and make use of diagonal covariance matrices in the Gaussian mixtures used to model the frame emission.

Figure 89: Two instances of the handwritten word "sees" showing how a same writer may use different allographs for the same grapheme (the first 's') even in the same word.

Relating the frame correlation, it seems convenient to distinguish two sources of variability in the observed signals:

EXTRA-SEGMENTAL factors remain fixed over the entire segment. Two examples are the chosen pronunciation of a speech sound (e.g. the allophone) or the speaker identity. Note that these features are related to the "auxiliary features" described in Section 3.1.15 (as a possible additional input data) which are, in principle, constant for the entire utterance. Not all extra-segmental sources of variation are due to auxiliary features since, for instance, the chosen allophone (or allograph, for the case of HTR) may vary from one segment realization to another one, as illustrated in Figure 89. Other factors such as the speaking rate are not necessarily constant but are at least constrained to a slow smooth variation;

INTRA-SEGMENTAL variations correspond to the differences observed in different realizations of the same unit by the same speaker, writer, etc.

Some SMs may address each of these sources of variation in a different way. For example, some SMs generate each frame from an independent outcome but the parameters associated to the frame emission distribution may follow a trajectory within the segment. In this way, the frame independence assumption might model intra-segmental variations while the parameters used to generate these frames would follow a trajectory from a distribution of trajectories to deal with extra-segmental variations, as in probabilistic-trajectory segmental HMMs [Holmes and Russell 1999]. Although these issues will be addressed later into more detail, it is convenient now to relate this with the distinction between parameters generated at the frame level from those generated at the segment level: The first ones would be associated to intra-segmental sources variability whereas those generated at the segment level would correspond to extra-segmental variabilities. The limitation of HMMs to model extra-segmental variability at the frame level means that, for instance, the use of Gaussian mixtures to deal with different types of speaker in ASR has inherent limitations since it is possible that one frame is modeled using the emission associated to one speaker while the next frame makes use of a component associated to a different type of speaker.

frame-level vs segment-level parameters

In general, the main way to tackle the frame independence assumption by most SMs other than conventional HMMs consists in taking the entire set of frames comprising the segment into account in order to obtain more accurate statistics, to use trajectory models (e.g. polynomial trajectories) or to rely on templates (their non-parametric counterpart). Trajectories and templates can be either duration specific or adapted to the actual segment length by means of warping or interpolation.

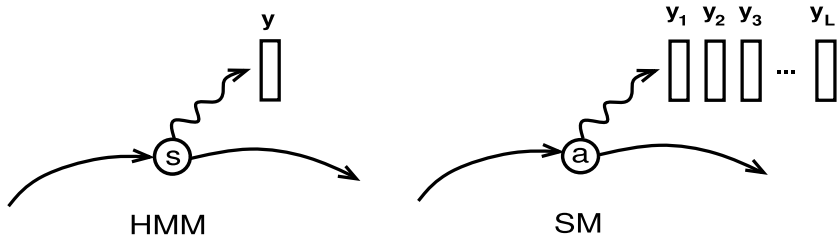


Figure 90: HMM and a more general form of SMs: On the left, HMM states produce one frame y at a time. On the right, the region a of the SM produces a variable length frame sequence $y_1, \ldots, y_L$ with a length L with a probability $p(L|a)$ (figure adapted from [Ostendorf *et al.* 1996; Fig. 1], but observe that we are using a different nomenclature).

7.2.3 Markov modeling of intra-segmental regions

The realization of a segment, given its underlying label, does not require the further division of the segment into smaller regions. For example, a sole trajectory or template may explain how the segment has been produced. On the other side, HMMs are miniature two-stage models which divide the segment into *one-frame regions*. There exists an intermediate possibility placed between both extremes: some SMs divide segments into regions which may comprise several frames each one, as is the general modeling framework described in [Ostendorf *et al.* 1996; Section 2.1]. These regions roughly play the same role as states in HMMs but emit frame-sequences, as illustrated in Figure 90.

Moore vs Mealy

The models depicted in this figure use the Moore representation of finite state machines, which is the most common choice[10] when representing HMMs.

Modeling region sequences in these SMs, as well as modeling state sequences in HMMs, is similar to dealing with sequence of underlying labels in the first stage of TSGMs. This is mainly described in Chapters 6 and 5. The Markov assumption of HMMs implies that they can model stochastic regular languages.[11] Since these languages can approximate any other stochastic language[12] and, since the number of regions in a segment is quite limited in practice,[13] we can conclude that the Markov assumption does not pose serious problems to deal with sequences of regions. Indeed, most papers describing the limitations of HMMs related to the Markov assumption put the emphasis on the weak duration modeling described below.

stochastic regular languages

10 A notable exception is [Jelinek 1998] where emissions are modeled in transitions, which corresponds to the Mealy representation. Nevertheless, both representations are equivalent (each one can be obtained from the other one).

11 The order of the Markov model does not affect that since a higher order model can be emulated by a model with a lower order one, at the expense of a size increase.

12 Depending on the metric used to compare languages. The probability mass of a stochastic language is concentrated on shorter sentences and the subset of sentences whose length is bounded with a given threshold is finite and hence regular.

13 Usually much lower than the number of labels in a sequence. As a rule of thumb, three HMM states for ASR and between 7 and 14 states for HTR. Indeed, most SMs found in the literature just describe a linear sequence of regions or, in some cases, a mixture of these sequences.

7.2.4 Weak duration modeling of HMMs

The term "duration" is used because most segment models have been historically described in the context of time series (and, more concretely, ASR). We believe that "segment length" is more task independent but "duration" and "durational" are long-established.

importance of duration

Phone duration plays a significant role in the comprehension of some languages: there are some word pairs in Finnish which are basically distinguishable by the duration of some of their phones [Hirsimäki *et al.* 2006]. Even for tasks where this distinction is not so determinant, it is important to obtain the most accurate models so that the expected duration of each type of segment should be as close as possible to the durations empirically observed on training data.

weal duration modeling

The number of frames emitted by a HMM state with a loop, before leaving this state, leads to a geometric or exponential distribution. More concretely, if the probability of staying in the same state is α, the duration distribution associated to this HMM state is:

$$p(\text{stay during } n \text{ frames}) = (1-\alpha)\alpha^{n-1}$$

which does not usually correspond to the empirically observed duration distribution of contiguous frames associated to the same emission in ASR (as well as in other) tasks.

HMMs may deal with duration modeling by properly choosing particular topologies, as described in the next section. Although HMMs could approximate, in principle, any duration distribution (at the expense of an increase in complexity of model topology), we will also see in the next section that some SMs try to overcome the duration limitations of HMMs by explicitly modeling duration distributions. These distributions can be either parametric or non-parametric.

Let us conclude this section by remarking that the problem of duration modeling can be alleviated during preprocessing by means of "speech rate normalization" [Pfau *et al.* 2000], in the case of ASR, or by means of "width normalization" in HTR (see Section 14.5.2).

7.3 IMPLICIT VS EXPLICIT DURATION

Segment models can be classified into two broad categories depending on how duration is taken into account:

IMPLICIT DURATION MODELS where the final segment duration is a *byproduct* of a process which generates the frame sequence;

EXPLICIT DURATION MODELS when the overall segment duration is chosen in a first place and a fixed length frame sequence of this particular length is generated afterwards.

HMMs constitute a clear example of implicit duration model since the duration is a result of the topology and transition probabilities (and it is independent of frame emissions).

The distinction between implicit and explicit is not perfect since it is possible to marginalize the duration distribution of implicit dura-

tion models in order to describe them as explicit ones, although this description is expected to be cumbersome and artificially complex: it is possible, for instance, to describe HMMs, *for finite sequences*, as if they were explicit duration models by precomputing the probability of each possible segment length (nevertheless, it would not be their natural description).

We can also consider two additional halfway or intermediate categories that, although we have not found them in the literature, seem coherent and useful to place some segment models:

IMPLICIT DURATION WITH EXPLICIT DURATION CONSTITUENTS, motivated form the fact that *our interpretation* of explicit duration models refers to the *overall* duration distribution, exclude models where the explicit duration distributions are applied at the state (or region) level, as in the case of hidden semi-Markov models described below. These hidden semi-Markov models, which have been traditionally coined as explicit duration (e.g. explicit duration HMMs), are rather considered here as implicit duration with explicit duration constituents;[14]

IMPLICIT DURATION WITH TIED DURATION PARAMETERS can be considered another refinement of implicit duration models where duration distributions of different regions depends on segment-level parameters. In this way, the actual duration of each region is no longer independent of the other from the same segment. We are not aware of this category in previous literature.

7.3.1 Implicit duration models (and topologies)

A particularity of duration in HMMs is that each visited state emits one frame at a time, which makes the segment length equivalent to the length of the path traversing hidden states. This equivalence no longer holds for SMs as those depicted in Figure 90 (right). The duration of these SMs is a result of the traversal of the hidden sequence of states and the duration distribution of each state.

Let us see how the state duration can be better specified in the HMM formalism before reviewing the topologies that are typically used for segment modeling in some ASR and HTR tasks.

Modeling state duration in HMMs

We have seen that a single HMM state with a loop generates an exponential duration distribution. The frames emitted by this state share the same frame emission distribution although each one corresponding to an independent throw. Since most HMMs are composed of several states arranged in more or less complex topologies, the overall duration is not necessarily exponential. This duration can even be described in a parametric form for some particular topologies. It seems quite simple, given any topology and the corresponding transition probabilities, to estimate the probability of each segment duration:

14 Things are not crystal-clear: HMM constituents emit one frame at a time: they are (trivial) explicit duration constituents. Likewise, an explicit duration model can be the sole constituent of an implicit duration model.

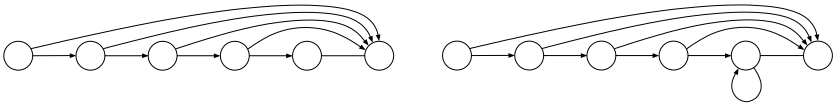


Figure 91: Fergusson sub-HMM topologies. Figure inspired by [Pylkkönen 2004; Figs. 4.3 and 4.4].

It suffices to jointly compute duration probabilities of segment prefixes ending at a given state. These values can be computed by means of dynamic programming from the same values on the previous duration of preceding states. Since the frame emission distributions of different HMM states can be tied, sub-HMMs with tied emissions can be constructed to approximate any desired duration distribution.[15] Fergusson topologies, illustrated in Figure 91, are often used to model such sub-HMMs. HMMs constructed using this approach are termed "super HMMs" or "expanded state HMMs" and the sub-networks associated to the same emission (used to emulate an state with a duration distribution other than geometrical) are known as "superstates".

An alternative to design complex topologies to approximate certain duration distributions consists in altering the basic HMMs to take the state duration explicitly into account, which violates the Markov assumption. This is the case of hidden semi-Markov models (HSMM) [Russell and Cook 1987; Yu 2010] where self transitions are replaced by an explicit state duration distribution. Indeed, the previously mentioned "expanded state HMMs" are often considered as a way to emulate HSMMs. Although most papers classify HSMMs as "explicit duration HMMs", in our criterion[16] only the states of HSMMs and not the HSMMs themselves are explicit duration models. HSMMs, excepting the trivial one-state without loops,[17] are considered here a less trivial form of implicit duration models that we have coined implicit duration with explicit duration constituents.

The extensions of HMMs with an explicit duration distribution of states require more complex decoders which have to consider the different durations in each state. This also happens with other general SMs such as explicit duration SMs. This explains the advantage of using HMMs with complex topologies (the expanded state HMMs described before). Another way to model the duration consists in applying an standard decoder to a simple HMM and to re-rank the scores of the resulting paths using better duration models [Juang *et al.* 1985; Rabiner and Juang 1986]. This approach is sub-optimal since durations given by the first simpler HMMs are not necessarily the same as those obtained by using one with more accurate duration models in the first place.

15 This approach no longer holds in the general case of SMs with frame sequence emissions at states (Figure 90 (right)) since the concatenation of two continuous trajectories is not equivalent to a sole trajectory of the same length.

16 We have dared to define our own criterion because there is not a consolidated consensus and because it seems, in our humble opinion, most coherent.

17 Implicit duration models require the concatenation of several (possibly the same) regions. Otherwise, any explicit duration distribution could be trivially considered as a implicit duration model with just one region.

Some common topologies (for implicit duration models)

It is clear that the HMM topology is relevant for duration modeling, but it also determines the sequence of types of frame emission. Although most algorithms for the estimation of HMM parameters adjust transition probabilities to take the duration into account,[18] the use of fixed HMM topologies impose a restriction on the family of achievable duration distributions. For these reasons, it is important to properly choose the HMM topology.

Unfortunately, there is no simple *well-established* method to joinlty determine the best HMM configuration and train it. The usual procedure usually consists in choosing a HMM topology family, which is parametrized by the number of states, and to determine the number of states either by means of a validation set or analytically from duration statistics. Let us review the most common HMM topologies found in the literature for ASR and HTR tasks. The following ones can be parametrized by the number of states:

ERGODIC HMMs corresponds to fully connected models (all transition probabilities are stricty positive), as depicted in Figure 92 *a)*. It is possible to transit from one state to any other one (including itself) at each moment. However, this generality does not necessarily mean that they are the better suited models for all applications. Indeed, ergodic HMMs have not proved useful in ASR or HTR tasks;[19]

LEFT–TO–RIGHT topologies (Figure 92 *b)*) correspond to acyclic graphs excepting the presence of loops. These models are characterized by an upper triangular transition probability matrix. They are very popular for modeling transient temporal processes as is the case of segment models.

Although they are also known as Bakis topology, let us remark that the expression "Bakis topology" is *overloaded* since it is also used to refer to the particular case of a left-to-right HMM *with skips* (Figure 92 *c)*), meaning that each state has only transitions to itself and to the next two successors [Bakis 1976];

LINEAR or strict left-to-right HMMs is a particular case of left-to-right when each state has a self loop and a transition to the following state, as illustrated in Figures 92 *d)* and *e)* depending on the presence of null transitions to skip some states.

There exists some variants of some of these topologies. For instance, the RWTH group on Aachen has reported some HTR works (e.g. [Kozielski *et al.* 2012; Doetsch *et al.* 2013; 2014]) using the topology described in Figure 92 *f)* which is, in essence, a Bakis topology where the frame emission of consecutive pairs of states[20] are tied.

18 Indeed, let us remark that assessment measures of sequence modeling (see Section 6.2) implicitly penalize duration mismatches. Let us also point out that some research groups specify fixed transition probabilities from the average duration of segments instead of training them (personal communicatio).

19 One exception seems to be the silence models as those reported in [Odell 1995; Section 6.1] which are three state HMMs with an ergodic topology excepting the initial and final transitions so that the model enforces a minimal duration of two frames.

20 Which they call "segments" to make this term even more overloaded.

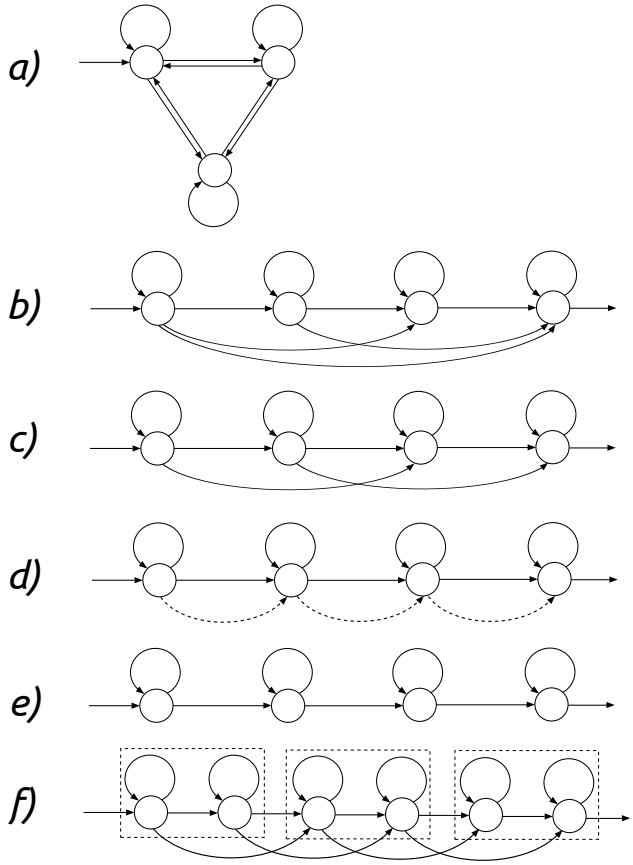


Figure 92: HMM topologies parametrizable by number of states: a) ergodic, b) left-to-right, c) Bakis or strict left-to-right with skips, d) strict left-to-right with null transitions, e) strict left-to-right or linear, and f) a Bakis topology based on pairs of tied states as described in [Kozielski *et al.* 2012].

ad hoc topologies

We can also found some HMM topologies which are not parametrizable by the number of states and hence can be considered "ad hoc". For instance, the IBM topology (Figure 93) was a popular HMM topology used in the baseline Sphinx system [Lee *et al.* 1990] and abandoned nowadays. Indeed, we can observe in the literature that most sophisticated HMM topologies were proposed shortly after the use of HMMs in ASR and were abandoned in favour of simpler topologies, as observed for instance in [Russel *et al.* 1990]:

> *In general, "rich" topologies with the ability to skip states gave poor results, due to their inability to successfully self-organise on data which was only labelled a t the sentence level. Based on these results it was decided to use a simple 3 state topology in which only self-transitions and transitions to immediately succeeding states were allowed.*

Several current ASR systems use a three-state strict left-to-right topology. Some works report the use of different models for some particular phonemes, as depicted in Figure 94 (right). Silence models are often constructed using a topology different from the rest of the models and,

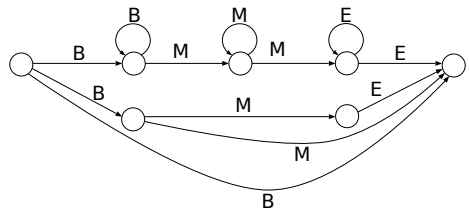


Figure 93: IBM HMM topology. Transitions are labeled by tied output distribution corresponding to beginning (B), middle (M) and end (E) of a phone. Lower transitions explicitly model durations of 1,2 and 3 frames. Upper transitions model 4 or more frames.

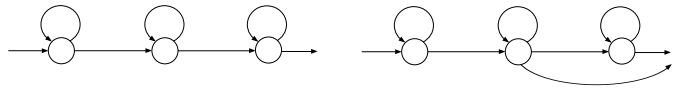


Figure 94: Two ad hoc ASR HMM topologies: (on the left) the classical three state left-to-right HMM used to model most phones and (on the right) a variant for optionally released stops (`dd`, `kd`, `pd` and `td`) (adapted from [Odell 1995; Fig 6.2b]).

in this case, two different models are often used: one for representing short optional silences between words (usually modeled with one state with a loop and an optional skip) and another to represent long pauses (which can be represented by a little ergodic model).

Most current ASR and HTR systems construct HMMs by joining linear sequences of quite elementary models, leading in some cases to some of the well known HMM topologies described before. Some of these elementary models have received a special nomenclature:[21]

FENONE is a simple HMM (as those depicted in Figure 95 *a)*) for representing sub-word speech units which can be obtained automatically from a training corpus [Bahl *et al.* 1988]. These models were used for HMM representing word models and the inventory of fenones was obtained by vector quantization without taking their phonetic representations into account;

MULTONE was a generalization of fenones by making a parallel combination of fenones;

SENONE [Hwang and Huang 1992] are one state topologies, as illustrated in Figure 95 *b)*. They share with fenones the fact of being automatically constructed in a clustering-like procedure which takes the entire sets of units into account. Senones are more widespread than fenones because they constitute states of phonetic based units. The idea of linear sequences of senones has been extended to tree and more general graph topologies as described in Chapter 10.

The problem of linear topologies is that they are not appropiate to model variants of segment realization (e.g. allographs in HTR). A quite general topology to deal with these variants is a parallel combination of linear models.

21 As a curiosity, these names have inspired other models sugh as "genones" [Digalakis *et al.* 1996] related to shared Gaussian codebooks, see Section 7.5.2.

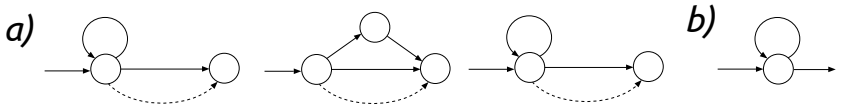


Figure 95: a) Some fenone topologies, b) senone topology.

Number of states

As mentioned before, the design of HMM topologies for a given task is usually limited to the choice of a family of topologies parametrized by the number of states and the selection of these numbers. Since these families are often variants of left-to-right models, the number of states is related to the average length of the generated segments. This value is usually chosen proportional to the average length of the training data. For instance, [Zimmermann and Bunke 2002b] investigates the use of three different schemes to optimize the number of states of linear left-to-right HMMs:

- each character model is assigned the same number of states, this allows a simple scanning of parameters guided by the results on validation data;
- the number of states may be different for each model. In this case, two different approaches are proposed based on the segment durations obtained from training data by means of a forced alignment:
  - **Bakis length modeling** where the number of model states is set to a given fraction of the average number of observations of the corresponding character;
  - **quantile length modeling** where the number of model states is set to a specified quantile of the corresponding character length histogram.

Let us remark that the number of states is also related to the number of types of frame emission probabilities, although this is an upper bound since some of them could be tied. Note also that certain HMM topologies impose a minimal duration for a given number of states, as is the case of linear topologies. This may pose a problem with too short segments which cannot be properly decoded.[22]

7.3.2 Explicit duration models

The probability of emitting a set of acoustic (in the case of ASR) observations a represented as a frame sequence of length l, for a unit of label u, when using explicit duration models, is:

$$p(a, l|u) = p(a|l, u)p(l|u) \tag{7.1}$$

As observed, these models require a segment duration distribution $p(l|u)$. This distribution can be non-parametric (e.g. smoothed du-

22 In most cases, the decoder is still capable of decoding by "stoling" frames from neighboring segments, although this seriously degrades performance.

ration frequencies obtained from a large training corpus) or parametric (e.g. Poisson, Gamma or truncated exponential distributions are among the ones reported in the literature).

Besides segment duration distributions, explicit duration models require a family of output densities $p(a|l,u)$, which is the subject of Section 7.4.

Topologies in explicit duration models

Some explicit duration SMs divide the segment into regions. The use of topologies in explicit duration models makes only sense in this case. Systems reported in the literature seem limited to linear sequences of regions although, in some cases, the parallel combination of linear sequences is also considered. As with explicit duration models, it seems also possible to specify sets of region sequences by means of finite state devices. The main issue is probably how to handle global duration constraints. To this respect, a mapping of frames into regions is sometimes performed either in a linear way or by means of warping, which is usually approached by means of dynamic programming techniques.

7.4 FRAME SEQUENCE EMISSION

The subject of this section is the estimation of the probability of observing a chunk of contiguous frames $a = a_1, a_2, \ldots, a_l$. We can usually assume that the segment length l is known in advance as in the estimation of $p(a|l,u)$ of Equation 7.1.

This section does not pretend to be a bibliographic survey of the existing frame emission models proposed in the literature but, rather, to give a *brief* scheme[23] capable of describing a space of design where these models could be structured in a way which makes sense to us. One of the reasons is that there are other surveys and it makes nonsense to repeat them: the reader is referred, for instance, to [Frankel 2003; Rosti 2004; Deng *et al.* 2006; Zen *et al.* 2007; Hu 2012] and references therein. These works usually cite [Ostendorf *et al.* 1996] as the main survey on the field but, since they are more recent, they propose their own review offering a classification emphasizing their points of view while discussing later contributions.

Relating previous work, one of the most problematic issues that can be probably found is the large amount of nomenclature plenty of clashes and subtleties. We are afraid of using a term with an exact meaning and prefer to rely on explaining the features when required. To this respect, we have to be aware of the fact that we are using the term "segment model" in a much more general way than other works since we are meaning, in general, models able to determine the likelihood of observing/generating a segment represented by means of a frame sequence. This also explains why we are not attempting the laborious and probably misleading work of providing a throughout

23 Such scheme is the aim of Section 7.8 which will take into account not only the models discussed here but also other possibilities as the use of discriminative classifiers at the segment level as well as model combination.

association of previous work with each possible model type that can be described in the proposed space of design. Our main aim is just to give a glimpse of the possibilities of these models to make the TSGMs of Chapter 4 more valuable.

The rest of this section is divided into two parts: first, the presence of optional, intermediate levels of description will be discussed. Later, some different ways to generate the frames (either for the observable or for an intermediate level of description) will be shortly addressed. Note that the particular case of generating a sole frame is delayed to Section 7.5. Observe that the emission of just one frame is not only a particular case but also a tool or mechanism suitable for the generation of frame sequences.

7.4.1 Intermediate levels of description

HDM vs others

According to [Deng *et al.* 2006], the main criterion to classify the existing SM types is at which "level" operate the models or, more precisely stated, how far they are from the surface or observable form. While some models directly explain how the observed frames are produced, other models known as Hidden Dynamic Models (HDM) make use of intermediate (and, hence, hidden) parameters usually related to articulatory parameters. Note that this work is specific of speech while we are trying to discuss these issues in a more general or abstract way. Relating speech, several levels of dynamics in human speech generation are distinguished in [Deng *et al.* 2006], ranging from the symbolic to the acoustic, and including the aforementioned articulatory related parameters (e.g. vocal tract resonance, formants and so on).

Although this classification seems mainly constrained to speech and aimed at describing the details of the speech production,[24] the idea of using a hidden intermediate representation is feasible in many tasks where TSGMs can be applied. Let us just observe that in handwriting, to put another example, there are also physical phenomena subject to motor planning, mechanical inertia and the like.

The idea of explaining the observed signal by means of a sole underlying linear sequence of (hidden and) discrete labels is the base and main feature (or limitation) of TSGMs. To this respect, observe that some phonological theories other than the linear phonology talk about non-linear or multi-linear sequences of asynchronous features. Although we have circumscribed our study to the basic assumption of a sole linear sequence of hidden discrete labels, we have also considered the possibility of establishing constraints between neighboring segments (the hierarchy proposed in Section 4.2.1). However, the use of an intermediate level of representation, as in HDMs, provides an alternative/additional way to deal with these constraints. This happens when the transformation of the hidden description into the observable form (e.g. "articulatory-to-acoustic mapping" in speech) may cross segment boundaries. Indeed, this mapping can be described as part of the generation of segments (as illustrated in Figure 96, on the left), or can be applied after a TSGM produces an output signal at the in-

24 We can remember here the dilemma of description versus explanation and the fact that an "aircraft do not need to flap its wings in order to fly".

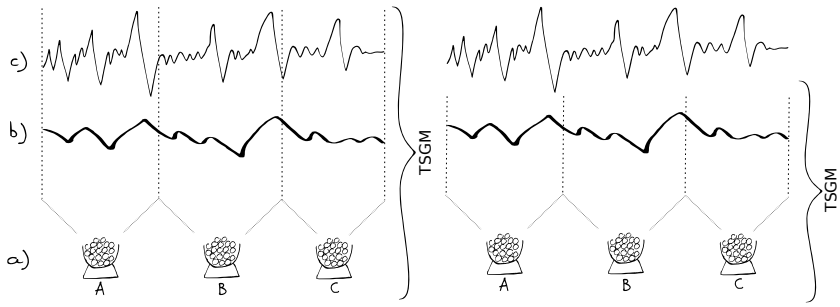


Figure 96: A set of hidden discrete labels a) is used to generate an intermediate signal/sequence b) that is used to produce the observable signal/sequence c). In the left, the second stage of the TSGM generates the observed signal by using the intermediate level of description as an internal detail. Contrarily, in the right figure, the TSGM produces the intermediate signal which is used to produce the observable one afterwards. The main difference is how the mapping from b) to c) (e.g. "articulatory-to-acoustic" mapping) is produced: in the first (left) case, each segment is produced independently and the continuity constraints have to be explicitly modeled whereas, in the second (right) case, these constraints can be partially tackled by the final stage.

termediate level of description (as shown in Figure 96, on the right). The capability of HDMs to allow the intermediate-to-observable mapping to cross segment boundaries explains why these models have also been coined as *super-segmental*. We can intuitively suppose that these flexibility is accompanied by an increase in the difficulty of decoding. Indeed, the high complexity limits the applicability of these models to rescore N-best lists or, perhaps, to work with lattices obtained in a previous search. We can also mention that many works using these models, such as [Zen *et al.* 2007; Hu 2012; Cai *et al.* 2015], are mainly focused on speech synthesis rather than on recognition.

HDMs may subsume other SMs

Let us observe that, since the intermediate-to-observable mapping of HDMs can be the identity function, SMs which do not use an intermediate level of description can trivially be a particular case of them. This is the reason why the generation of sequences, no matter if they are related to the intermediate representation or to the observable or surface form, will be joinlty tackled in the same subsection.

Relating the kind of intermediate-to-observable mapping functions that can be envisaged, some works explicitly dealing with them, such as [Hu 2012; Section 2.4.4], distinguish between linear and non-linear mappings, being the use of connectionist techniques a notable form of non-linear mapping. We can also note that the inclusion of augmented or dynamic features in the form of delta parameters (e.g. first and second derivatives), widely applied together with HMMs and often criticized, can also be explained as a (deterministic) mapping from an intermediate level (the original frames) to the observable ones (the same frames with augmented features), as depicted in Figure 97. The trajectory model of [Zen *et al.* 2007] tries to take this deterministic mapping into account, hence resembling HDMs.

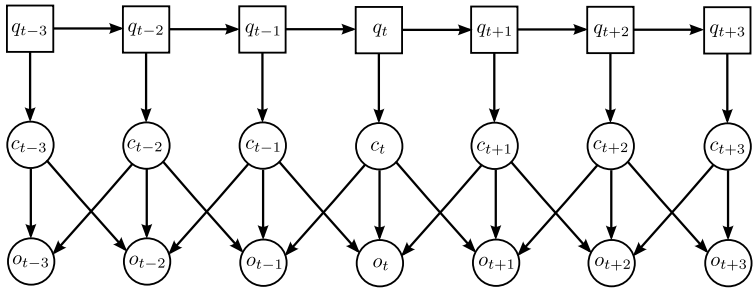


Figure 97: Graphical view of augmented features illustrating a first layer of features and the augmented features (e.g first derivatives).

7.4.2 Trajectories, state-space models and non-parametric templates

Most frame sequence emissions models reported in the literature describe a trajectory that can be subject to random variations. The simplest from of trajectory is a constant one. This trivial particular case, when the trajectory is placed on an intermediate level of description, allows the generation of a set of observable frames that, even if they correspond to independent throws of a frame emission model, may account for extra-segmental sources of variation (for instance, when the values of the constant trajectory are not fixed).

It is possible to define trajectories in a parametric explicit way (as a function of a frame index parameter) or in a non-parametric way. Polynomial functions have been reported in the literature (e.g. trended HMMs). Their coefficients can be fixed or stochastic.[25] Non-parametric trajectories are often given for a reduced set of segment length and often extended to other ones by means of warping.

This classification is not exhaustive since we can define trajectories without an explicit frame index by modeling the state evolution dynamics. A quite general framework for describing these dynamics are state space models [Frankel 2003; Rosti 2004]:

$$y_t = h(x_t, \epsilon_t) \tag{7.2}$$

$$x_t = f(x_{t-1}, \ldots, x_1, \eta_t) \tag{7.3}$$

Function f defines the evolution of the internal state process x whereas h is used to obtain the observation from this internal state. Observe that x and y may have a different dimensionality[26] and that the observation only depends on the current state vector.[27] Let us remark that, although piece-wise constant state evolution is quite common (e.g. HMMs), the evolution of x may be continuous. The observation noise ϵ_t is used to represent the intra-segmental sources of variation whereas the state noise η_t is used to model the uncertainty in the evolution process of the internal state. Another notable particular case are those where f and g are restricted to be linear and, more particularly, the case of linear Gaussian models where observation and state noises are Gaussian distributed.

25 Observe that the constant trajectory discussed before is a particular case.
26 The dimensionality of the observation is usually higher.
27 They can be converted into this form when observations depends on a finite length past.

7.5 FRAME EMISSION

Frame emission is not only a notable particular case of the previous frame sequence emission which deserves a sole section by itself, but also a mechanism that can be used in the generation of frame sequences. For instance, continuous HMMs often use GMMs to model the emission of observed frames, and the parameters of these mixtures are fixed for each HMM state, but it is also possible to use Gaussian models whose parameters from trajectories [Ostendorf *et al.* 1996].

When the frame emission is not part of a frame sequence emission driven by an underlying intermediate level, as in HDMs, we are usually dealing with HMMs. Relating HMMs, a huge amount of literature is devoted to the tiniest specific details regarding the different types of emissions and how to better train, estimate and combine them. Also, many different types of HMMs proposed in the literature, usually with a showy nomenclature, are, in essence, classical HMMs which only differ at some specific points.

As stated before, the structure of this sub-section does not pretend to completely classify this vast amount of literature nor to provide a general scheme where, as a generative grammar, new types of frame emission fitting this scheme could be envisaged. These aims would undoubtedly surpass our resources, so only a glimpse towards this direction is provided.

classification by observed variables

Before presenting such an scheme, let us mention that HMMs are traditionally classified, depending on the nature of the observed variables, as follows:[28]

DISCRETE when observed variables take values on a discrete space;
CONTINUOUS DENSITY when observed variables are continuous.

7.5.1 From discrete features

Discrete non-parametric models

Discrete observations, unless otherwise stated, are usually modeled by means of non-parametric models. They offer some advantages: On the one side, they do not require any a priori assumption over the type of distribution; on the other side, they are very easy to implement (a simple table lookup). Parameters are estimated from the training data by counting during the EM process, although they suffer sparseness problems when the discrete space is too large. This problem is specially relevant for those tasks which are continuous in nature but a vector quantization process has been applied to be able to use discrete output probabilities. In this case, the information lost during the quantization process is another drawback to be taken into account. Nevertheless, the low computational cost, the advantages of non-parametric models and the fact that quantized parameters can be transmitted with a low bitrate makes them suitable for client-server systems (e.g. ASR on poorly-resourced handheld devices).

28 We have avoided, in this classification, the case of "semi-continuous HMMs" which are sometimes placed at this level of the hierarchy in other works. They are not a different type of emission probability but just particular types of emission probability for continous density HMMs described below.

Discrete parametric distributions

Although it is possible to mitigate the sparseness problem of non-parametric models by means of smoothing techniques, it may be preferable to use parametric models in some cases. Among the most well-known discrete probability distributions we can find the binomial, the negative binomial, the geometric, the Poisson and the Bernoulli distributions, which can be employed as components of discrete mixture distributions.

multivariate discrete

In some cases, observed frames can be factorized as a tuple of discrete values, which can be modeled by means of multivariate discrete distributions and their corresponding mixtures. An example is the representation of a vertical slice of a black and white image to perform offline HTR. In this case, each pixel column is composed by a fixed[29] number of binary values (pixels) and it is possible to model them by means of a mixture of Bernoulli distributions [Giménez and Juan 2009].[30] Another example is when a continuous signal is quantized by using subvector quantization, which basically consists in partitioning the observed frames into subvectors which are independently encoded with different codebooks.[31] Each observed frame is associated one index for each subvector and discrete-mixtures can be used to model the emissions [Tsakalidis *et al.* 1999].

Observe that it is also possible to use some models in principle devoted to continuous signals (e.g. MLPs) for discrete observations.

7.5.2 From continuous features

Gaussian Mixture Models

Gaussian Mixture Models (GMM) can approximate any continuous density with arbitrary accuracy provided that a sufficient number of Gaussians is used. This is why they are one of the most widespread type of emission in HMMs and the literature relating the different types of models and techniques for parameter estimation and evaluation is overwhelming.

Covariance matrices are normally assumed to be diagonal in order to limit the number of model parameters, although this also assumes that different components of observed frames are uncorrelated, which can be achieved by means of decorrelation techniques in the feature extraction process. Another very common technique to reduce the number of parameters consists in tying some parameters[32] or even entire components of mixtures.

A typical large vocabulary ASR with context dependent units may require thousands of types of emissions composed by mixtures of up to 64 Gaussians. The most typical ASR parametrization uses the first 12 MFCCs along with normalized log-energy which, with the first and

29 A way to normalize the image height is assumed.

30 Nevertheless, HTR preprocessing is described in detail in Chapter 14, where we clearly defend the use of gray level and the use of continuous features which, to our knowledge, lead to much better results than directly processing black and white images.

31 It is even possible to optimally assign bits to different subvectors in order to optimize the resulting WER [Digalakis *et al.* 1998].

32 It is quite usual, for instance, to share the covariance matrix among several Gaussians.

second derivatives, leads to 39-dimensional feature vectors. Not surprisingly, the computation of these likelihoods takes a significant proportion of the computational load.[33] This cost explains the existence of a wide variety of fast likelihood computation techniques. They rely on a combination of different tactics since it is possible to optimize the models (e.g. by tying different mixture components) or to speed up the computation for a given model either by means of machine optimization (e.g. the use of certain special Single Instruction Multiple Data (SIMD) instructions of some processor architectures or the use of graphics processing units (GPUs)) or by means of algorithmic techniques (e.g. gaussian selection, described below). Note that, although these tactics usually try to reduce the computational load, it is also desirable to reduce the memory footprint. According to [Chan *et al.* 2004], algorithmic optimization techniques can be applied at different levels, so that lower ones may provide information to upper ones (from upper to lower level):

- frame-layer: the entire frame observations could be skipped when the frame is very similar to the previous one, the previous GMM scores can be reused [Woszczyna 1998; Section 5.3];
- GMM-layer: when the system may ignore some GMMs. For instance, some systems compute GMMs for context independent models and only compute the context dependent counterparts of those which survive a pruning stage;
- Gaussian-layer: most fast evaluation techniques rely on the assumption that only a few Gaussians will dominate the score of the mixture, so that selecting this subset for a given particular frame (hence discarding the others) accelerate the computation without a significant loss in accuracy. These techniques are known as Gaussian selection;
- component-layer: when the feature vector is divided into different subspaces so that the previous Gaussian selection techniques can be applied to each one independently.

Semi-continuous HMM

HMMs are traditionally classified, in terms of the nature of observed variables, as either discrete or continuous. By its name, semi-continuous HMMs [Huang and Jack 1989; Riedhammer *et al.* 2012] seem to be in the middle, but they are continuous. The confusion comes from the fact that they are sometimes described as an improvement over discrete distributions: they are also based on the use of vector quantization and a codebook, and the improvement consists in representing each cluster by means of a (Gaussian) continuous density function instead of the centroid in order to avoid quantization errors. Each type of emission probability is described as a mixture of Gaussians taken from a large set so that these density functions are shared. That is why they are also known as tied mixture models.

Note that the components of the tied mixture are not necessarily Gaussian and, indeed, they can be obtained from posteriors (applying

33 Between 30% and 70%, according to [Gales *et al.* 1999], although this result is surely outdated.

Bayes theorem in a similar way as described below) to lead to tied posteriors [Rottland and Rigoll 2000].

Relating the use of semi-continuous HMMs based on Gaussian models, it seems to exist a continuous spectrum of techniques, depending on the amount of tying and sharing, ranging from those described as GMMs to the ones described here. Besides, GMMs are very related to a particular case of Radial Basis Functions (RBFs) with Gaussian kernels (RBFs, have been used as emission of HMMs [Ney 1991; Singer and Lippman 1992; Gilloux *et al.* 1995]) and is also related to mixture density networks [Bishop 2006; Section 5.6].

7.5.3 Estimation from frame posteriors

The estimation of frame-wise segment posteriors is a problem type by itself which has been discussed in Section 4.4.3. It was also described, in that section, as an alternative way to tackle the problem of decoding. Now, our aim in this section is how to use frame-wise posteriors, computed from local features around a given frame, to obtain the observations of a HMM.

We depart from a discriminative model capable of estimating the probability $p(q|x)$ of each possible type of HMM emission q given the set of features x extracted from the current frame. Observe that this is not the same as the HMM emission $p(x|q)$ which is, rather, the likelihood that this frame has been emitted or observed given the type of emission associated to the HMM state.

The technique usually proposed to solve this mismatch (see, for instance, [Bourlard and Morgan 1994] and references therein) consist in applying Bayes theorem as follows:

$$p(x \mid q) = \frac{p(q \mid x)\, p(x)}{p(q)} \tag{7.4}$$

We can ignore the value $p(x)$ since it is the same for all likelihoods associated to the same frame and it can be ignored in the maximization involved during decoding, leading to *scaled* likelihoods:

$$p(x \mid q) \propto \frac{p(q \mid x)}{p(q)} \tag{7.5}$$

Relating the class priors $p(q)$, they can be estimated from the relative frequencies of the states associated to them (there can be shared emissions) that, in turn, can be obtained by means of a forced Viterbi alignment of the training data. This procedure is interleaved with the learning of posteriors in a EM procedure until some stopping or convergence criterion is reached. Although the estimation of models is not the issue of this chapter, let us point out that some details of this training procedure, applied to HTR tasks, can be found in Chapter 15.

In some works [Doetsch *et al.* 2011; Tüske *et al.* 2012] an additonal exponential term κ is included in the class priors to control the sharpness:

$$p(x \mid q) \propto \frac{p(q \mid x)}{p(q)^{\kappa}} \tag{7.6}$$

This term can be optimized using a validation set.

HMM/ANNs

The discriminative models reported in previous literature, used for estimating $p(q|x)$, are usually ANNs either in the form of:

- feed-forward models (MLPs)) [Bourlard and Morgan 1994];
- recurrent neural networks (RNNs) [Robinson *et al.* 1996]; or
- radial basis functions (RBFs) [Singer and Lippman 1992].

Let us remark that, although the terms "hybrid HMM" or "hybrid HMM/ANN" are often used to describe these models, we have to be aware of the fact that "Hybrid HMM/ANNs" is an overloaded expression which may usually refer to the use of ANNs in conjunction with HMMs no matter if they are used for obtaining frame posteriors. Indeed, the different uses of ANNs in this context is quite diverse:

- the approach based on the computation of scaled likelihoods from frame-posteriors described before. Nevertheless, let us remark that techniques other than ANNs exist for computing frame-posteriors (for instance, by means of SVMs [Stadermann and Rigoll 2004; Krüger *et al.* 2005]);
- ANNs can be used to directly estimate probability density functions [Trentin and Freno 2009];
- the use of ANNs to perform the role of a vector quantizer (clustering) in order to use discrete HMMs. For instance, the use of Learning Vector Quantization (LVQ) [Kohonen 1988] has been applied in the HMM/LVQ hybrid algorithm of [Katagiri and Lee 1993]. In a related work [Rigoll 1994] a neural network trained in an unsupervised mode, using the MMI criterion;
- ANNs can be used to predict the next frame given the previous ones like a non-linear autoregressive model. It is possible to obtain HMM observations by comparing the errors obtained by different ANN predictors, each one trained on a different type of emission [Tebelskis 1995; Chapter 6];
- although ANNs can be used to compute frame posteriors, they can, in turn, be used as features. At least two different approaches have been described in the literature, as is the case of [Bengio *et al.* 1991] or in the more recent *tandem* approach [Hermansky *et al.* 2000] where the posteriors are used as features for a HMM based con GMMs. Several variants exists and we can find that, in some cases, the original features are used together with the computed posteriors. The use of decorrelation methods (e.g. PCA) is also a common practice, and, in the case of ASR, to use very large windows of band limited spectral densities (TRAPs) [Hermansky 2003];
- the frame posteriors computed by ANNs can be used as labels. Using this interpretation, the observation score (log-likelihood) can be replaced by the KL-divergence between the vector of posteriors and a distribution associated to each type of emission, as proposed in [Aradilla *et al.* 2007];
- it is possible to obtain features other than frame posteriors from a ANN, as is the case of bottleneck features [Grézl *et al.* 2007].

7.5.4 A general scheme of frame emission

This previous distinction between discrete and continuous HMMs is not exhaustive since other types of frames can be envisaged, for instance, frames composed by a mix of discrete and continuous features and, in general, frames not restricted to finite length feature vectors but adopting the form of structured data, as is the case of sequences,[34] trees or even graphs.

Likewise, the description of the most common types of models for the emission of observable frames in HMMs is far from covering the proposals that have been described in previous literature. As with SMs in general, we are not aiming at providing such a bibliographic survey but, rather, to offer a short overview in the form of a scheme where it is quite easy to place previous works and, hopefully, novel proposals yet to come.

In order to propose this general scheme of frame emission, it is convenient to describe not only different types of frames but also other entities such as the desired (scaled) likelihoods produced as a result of the frame emission estimation process. Our main aim consist in placing these entities as nodes of a hypergraph linked by hyperedges whose semantics is that there is a way to obtain an element from possibly several (and, hence, the use of hyperedges) sources. A first attempt on this direction is depicted in Figure 98. The entities described in this figure are:

1. **DISCRETE FEATURES** can be of this form due to the task or can be obtained from continuous features by means of some type of quantization (k-means clustering, LVQ, etc.). Also, labels can be trivially interpreted as discrete labels;
2. **CONTINUOUS FEATURES** which has been addressed before, can be used with GMMs (which are a particular case of exponential models). They can also be converted into discrete ones or can be used to estimate (scaled) likelihoods;
3. **STRUCTURED FEATURES** are not very common, but at least we can find a HTR technique [Fischer 2012] (see Figure 53) where the text line to be recognized is converted into a graph which is traversed by a sliding window to obtain observation graphs. The use of graph distance techniques applied between the observed frame and a set of templates may convert these observations into numerical values (scores which, in turn, can be interpreted as features);
4. **(STATE) POSTERIORS**, not to be confused with frame posteriors, are computed from a (previous) forward-backward stage and, in contrast to frame posteriors, may have a more global interpretation whereas frame posteriors are computed from more local

34 Even if the observed frame is a fixed dimension feature vector, it is possible to process this frame as if it was a sequence. An example of this approach is the HMM2 described [Weber *et al.* 2001] where frames are generated by another HMM traversing the frame. Another example is known as "non-symmetric half plane" Markov chain [Jeng and Woods 1987] where observations are column pixels which are estimated by a Markov Random Field. Note that, as far as these frames have a predetermined number of parameters, this approach is rather a technique for emission estimation rather than a novel type of emission.

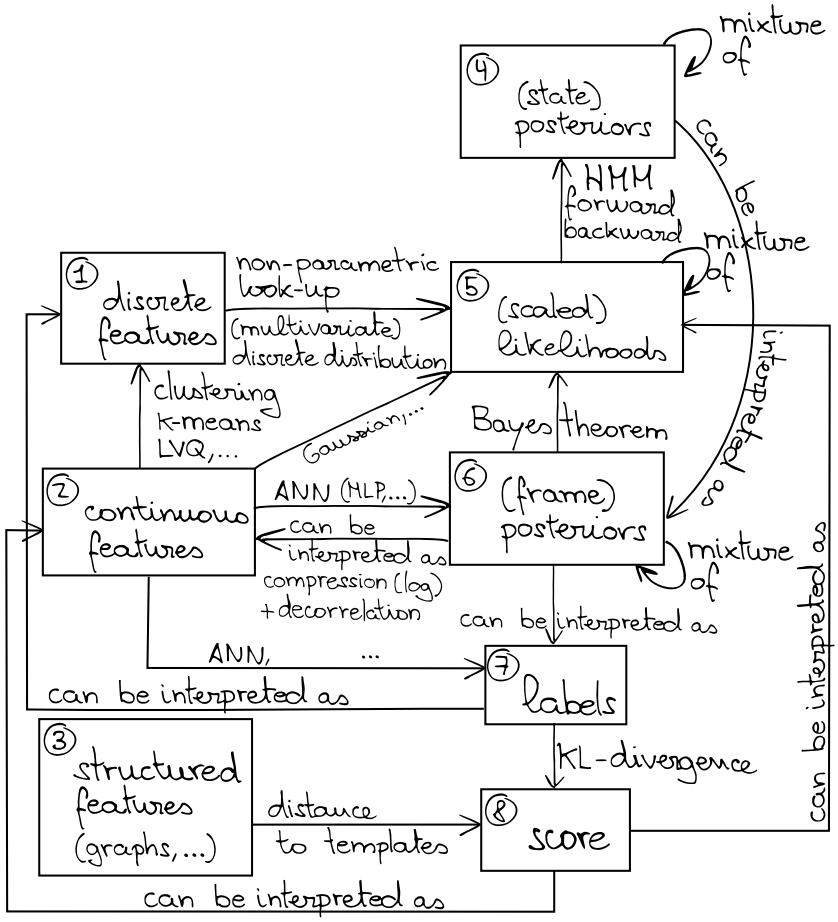


Figure 98: A general scheme of frame emissions.

features. Nevertheless, two remarks are worth considering: on the one side, it is possible to use them as if they were frame posteriors and, on the other, the estimation of frame posteriors by means of, for instance, bidirectional RNNs, may also have, in principle, a global scope;

5. **(SCALED) LIKELIHOODS** of emitting the observed frame given the class of emission is the actual value we are seeking from this scheme;
6. **FRAME POSTERIORS** represent the probability $p(q|x)$ of a class q given the observed frame x and has been addressed in previus section where we saw that they can be used at least in three different settings: to obtain scaled likelihoods, as features and as labels;
7. **LABELS** represent, in principle, class values from a categorical nature, although they can be interpreted as discrete features;
8. **SCORE** is a numerical value which does not necessarily correspond to a (possibly compressed and/or scaled) probability (posterior, likelihood).

Besides nodes, we can also see in Figure 98 several hyperedges showing how the different types of values can be related. Some of these relations have already been described and others are self explained. Some worth mentioning relations and observations are:

- (state and frame) posteriors and likelihoods can be combined in mixtures;
- when frame posteriors are used as features (as in the tandem approach [Hermansky *et al.* 2000]) it is usual to apply decorrelation techniques;
- it is also possible to obtain an score from labels by means of the KL-divergence [Aradilla *et al.* 2007].

The role of deep learning in frame emission

We cannot conclude this brief section without talking about the huge impact of deep learning on recent years in this setting. Much has been written relating these models, so the short answer is that they fit in this scheme *as is*: They are quite orthogonal to this classification since, in essence, these techniques replace others already existing: discriminative classifiers based on deep models to obtain frame posteriors [Mohamed *et al.* 2011] interface with HMMs in the same way as was previously done. Also,, deep auto-enconders can be used instead of more classical decorrelation techniques (e.g. bottleneck features [Grézl *et al.* 2007] have their deep counterpart in [Sainath *et al.* 2012], LVQ seems to have its counterpart in [Huang *et al.* 2014], and so on). This does not mean, in any way, that all deep learning techniques can fit into this scheme, only that the deep related techniques related to it does not make a significant change in the scheme because it is probably quite coarse. We can mention the existence of some end-to-end approaches based on deep learning related with models described in Chapter 4 as alternative to TSGMs, as is the case of [Hannun *et al.* 2014] based on CTC classification described in Section 4.4.3.

7.6 CONTEXT DEPENDENCY

Context dependent units have been briefly discussed in Section 2.8, for the case of ASR, and Section 4.2.1 where a hierarchy for the second stage of the two stage generative model distinguished the influence of neighboring labels in the generation of a given segment. Although this influence depends on the task at hand, it constitutes an standard in ASR systems nowadays since most acoustic models are based on triphones. The importance of context dependent units may is also related with the segment estimation techniques and with what are considered the basic units of our problem:

- some SM types take into account additional frames from the neighborhood. In particular, hybrid HMM/ANNs are based on ANNs to estimate frame emission probabilities and they usually receive a left and right context comprising several frames that may include frames from neighboring segments. Although it has been reported the use of context dependent HMM/ANNs, the context independent counterpart behaves quite well;
- some ASR decoders use words, syllables or demi-syllables instead of phones. In this case, a lack of context dependency would be alleviated or limited since some co-articulation features are tackled inside the segment and also because of the fact that phonetic information is integrated at syllabic timescales (see Section 2.8).

specificity

The following nomenclature is not task independent and seems tailored for ASR. We have maintained it because it is widely adopted and there are not task independent alternatives (remember Figure 5). Relating the levels of specificity we can find:

- monophone models, also known as context independent;
- biphone (conditioned inmediatly to the left or to the right), not to confound with demiphone which refers to a single model generating the second half of a phone with the first half of the next one;
- triphone, quinphone, and, in general, arbitrary context. Despite there is no limit to how far the context can be taken into account, there are several good reasons to limit this value, such as: the course of dimensionality, data sparseness or an increase in decoding complexity.

clustering contexts

A problem of context dependent units is the large amount of data required to train the large number of models which have an highly specific context. This is the reason why most inventories of context dependent units usually contain less models than the number of allowed combinations since many works propose to grouped them into clusters and, also, to use several levels of context specificity. This can be performed in several ways:

BACKING OFF when there is not enough data to reliably train a model, it is possible to back-off and use a less specific model for which there is more data;

Figure 99: The use of allophones/allographs may depend on neighboring labels. In this example, the particular shape of 'r' at the beginning of the word may be due to the fact that the following letter is 'e'.

SMOOTHING instead of relying on a less specific model, it is possible to smooth the parameters of the more specific model with those of a less specific one;

SHARING models or parts of models between different contexts. This can also be based on clustering.

Sometimes, the same recognition system can use both context independent and context dependent units. An example is the fast match look-ahead technique described in Section 10.6.2. It is also possible to construct context independent models by aggregating the context dependent emission probabilities [Linarès *et al.* 2007; Section 2.2].

position dependency

Besides the context given by neighboring labels, some authors (e.g. [Hwang *et al.* 1996] or [Sixtus 2003; Section 6.4.2]) have reported the influence of the position of the label within the word.

We can mention (mainly for completeness), in the case of speech, the use of specialized models for gender and age and even for speech rate, which are not related to the neighboring context but, rather, influence all the segments no matter their labels. These features seem more related to parametric HMMs and external variables mentioned in Section 4.2.2.

allophone dependency

Other feature which, in our opinion, seems related to context dependency is the possibility of having several variants of the segments associated to a given label, as is the case of allophones (different realizations or variations of phones) in ASR or allographs in HTR (depicted in Figure 12). We believe that the concrete choice of these allophones may depend on the context and, of course, on the particular choice of allophones to model neighboring labels (otherwise stated, they may be correlated), as illustrated in Figure 99.

demiphones

We have also seen that a segment model can be broken into independent parts, as is the case of demiphones [Mariño *et al.* 1997; Menéndez-Pidal *et al.* 2007], depicted in Figure 100(2), where the first part of the model only depends on the left context and the second part on the right one. Some works [Menéndez-Pidal *et al.* 2007] suggest that the influence of previous and following phones is not symmetrical and, hence, they propose topologies with a different number of states at each demiphone side. A type of context dependency related to demiphones are the Stationary-Transitional Units [Gemello *et al.* 1997] where transitions and the internal stationary part are modeled independently, as illustrated in Figure 100(3).

halfway units

An unconventional type of context dependency is the use of what we could coin as "halfway units" where the segment model is related to the final part of a label and the beginning of the next one. Otherwise stated, the transition not broken into the parts corresponding to each different label but, jointly beginning to both of them, as shown in

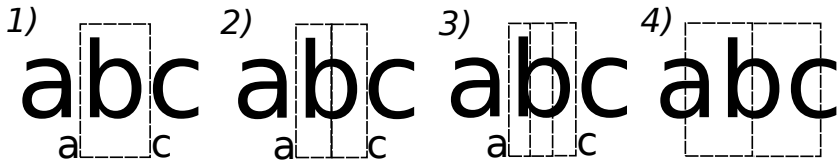


Figure 100: Some context dependent units types: 1) triphone ${}_{a}b_{c}$ representing label b with a given context, 2) a demiphone where the left part is not dependent on the right context, 3) Stationary-Transitional Units [Gemello *et al.* 1997] and 4) *halfway* units where only transitions between units are modeled instead of the actual units.

Figure 100(4). Although they have been used in speech synthesis, we are not aware of their use for decoding.[35]

7.7 ALLOWING SEGMENT-BASED OBSERVATIONS AND DISCRIMINATIVE MODELS

As explained in Section 4.4 and remarked at the introduction of this chapter, segment-based observations[36] are usually based on discriminative approaches and do not fit well with TSGMs since it is not possible to properly compare hypothesis associated to different segmentation paths, as illustrated in Figure 48. Another example is that, when they are applied to incorrectly segmented regions, they have to distribute the entire probability mass to the set of labels and have no mechanisms to indicate that this segment is not correct.

Observe that, although the use of discriminative techniques used to obtain segment-based observations is similar to the use of frame posteriors, the estimation of scaled likelihoods from these frame posteriors (described in Section 7.5.3) was possible by ignoring $p(x)$ in Equation 7.5 to lead to Equation 7.4 and this cannot be extrapolated to the case of segment-based observations.

A method to allow the use of discriminative models applied to frame sequences in the generative framework is proposed in this section. The technique has two different restrictions: on the one side, the features used by the discriminative models, even if they finally have the form of a fixed-length feature vector associated to the entire segment, are obtained from a frame sequence. On the other side, they cannot be used in isolation since their role is to re-distribute the probability mass previously obtained by a former generative model as follows:

$$p(\text{segment}|\text{unit}) = \frac{p(\text{unit}|\text{segment})\, p(\text{segment})}{p(\text{unit})}$$

35 They can be emulated, to some extent, by means of context dependent diphone models.
36 Those where a fixed-length feature vector is obtained for each segment no matter its length.

where $p(\text{unit}|\text{segment})$ is estimated by the discriminative model whereas $p(\text{segment})$ can be estimated by marginalization of the generative model over all unit types:

$$p(\text{segment}) = \sum_{u \in \text{units}} p(\text{segment}|u)$$

Let us remark that the generative model does not usually estimate $p(\text{segment}|\text{unit})$ but a scaled likelidhood where the scale is realted to $\prod p(\text{frame})$ where frames are those comprising the segment.

Relating the generative model applied to the segments, we can also remark that, when this estimation is obtained by means of HMMs, although they are often computed with the Viterbi approximation it seems better to use the forward step in this case.

An extension worth considering is to combine the result of the previous discriminative reranking with the original generative model. Although this can be seen as a particular case of model combination discussed in the next section, the cost of this combination is free in this particular case.

Restricting the discriminative classifier to a subset of units.

We can also envisage the application of this technique to discriminative models trained to take into account a reduced subset of units while other units are not classified and tackled with the reject option.

This idea has been applied in Section 15.5 for an isolated HTR task. As described in this section, a ANNs has been trained to perform a classification of word images at the word level (holistically) for a subset of short words.

The rationale of this approach is the empirical observation that short words had a significant contribution to the error rate.[37] A particularity of the application of this use of discriminative models to rerank generative ones is that it is usually applied at the word level.[38] When this technique is applied in continuous HTR, the use of an oversegmenter at the word level (as described in Section 14.7 and used in the experimentation part in Chapter 15) makes it computational effective. In any case (isolated or continuous), since it has been applied on short words, a criterion is required to determine whether the hypothesized segment (whose word identity we cannot know) is short enough to apply the holistic discriminative classifier or not. As mentioned before, this classifier must consider the reject option to deal with those words out of the selected subset. It remains to empirically study the possibility of mismatches since the combination with the discriminative model is not applied in all instances but only for some segments.

37 This seems somewhat related to the fact that some ASR systems have explicit HMMs to model function words.

38 This is the reason why this feature is not necessarily implemented as a segment model but, rather, as a post-processing in other places such as, for instance, in lexicon estimators (described in Chapter 10).

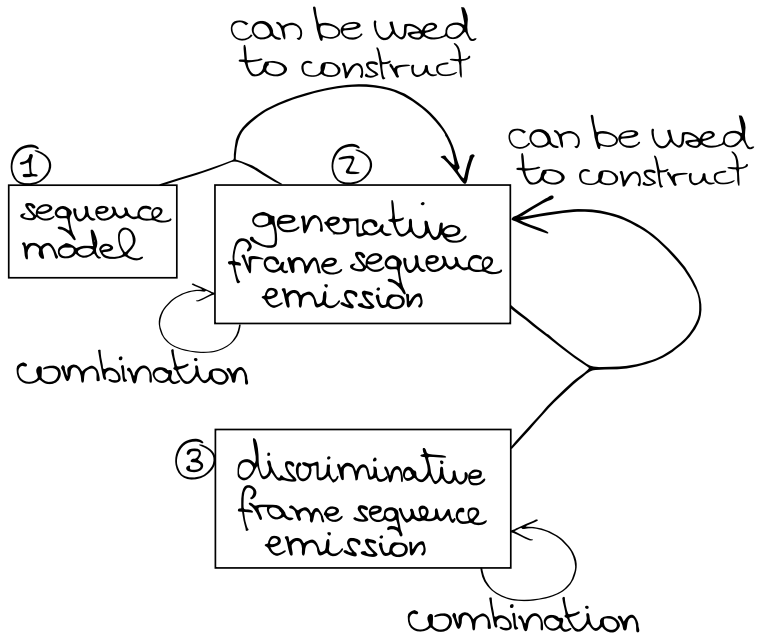


Figure 101: A general scheme of SMs illustrating the relationship between generative segment models, discriminative segment models and models of sequences (a.k.a. language models).

7.8 SOME STEPS TOWARDS A GENERAL SCHEME OF SEGMENT MODELS

This section is quite short since we can take profit of what has been previously discussed through this chapter. We only need to complenent previous content by acknowledging that the material covered in Sections 7.3 and 7.4 does not suffice to explain all possible SMs. Some examples not covered before are the combination of models at the segment level and the combination of generative and discriminative models of previous section.

A simple scheme joining all these ideas is depicted in Figure 101. As with the general scheme of Section 7.5.4, the underlying idea is represented by means of an hypergraph where nodes represent entities and hyper-edges indicate that it is possible to obtain an entity from other ones. The basic entities depicted in Figure 101 are:

1. **SEQUENCE MODELS**, described in Chapter 6 to the case of lexical units and called LMs, can be used in this context to describe sequences of regions in which a segment can be divided;
2. **GENERATIVE FRAME SEQUENCE EMISSION MODELS** is what we have generally called "segment model". It is, so to speak, the main goal of the scheme. But it is also the (parcial) origin of some hyper-edges since the emission of a frame sequence can be part of another SM (for instance, to model a region);
3. **DISCRIMINATIVE SEQUENCE EMISSION MODELS** can be combined with a generative model as explained in Section 7.7.

The high level of abstraction explains the lack of details where, for instance, we can observe that entire families of SMs and other different type of models are supposed to be included in these nodes. Indeed, each part of the scheme has been addressed elsewhere. We can finally observe that pure segment-based observations are not present in the scheme for the reasons explained before.

7.9 SUMMARY AND SOME CONCLUSIONS

This chapter has addressed the second part of the TSGM, namely: generative models for the joint segmentation and classification of sequences described in Chapter 4. Models that does not fit this scheme have been left completely aside, as is the case of SMs based on segment observations.[39]

SMs boost the relevance of TSGMs

Although SMs are not our main focus, the possibility of using SMs other than HMMs is one of the key factors and strengths of some DAG generation and decoding algorithms discussed and proposed in this work. These SM types are, indeed, the main reason why TSGMs are a *proper superset* of HMMs and we are obviously very interested in enhancing them. It is clar, from this point of view, that, although we have tried to provide an overview of SMs, this chapter has not pretended to be a review of the vast amount of SM types described in the literature.

chapter structure

The structure of this chapter has been a trade-off between what seems a logical structure (from our humble point of view) and the practical relevance of these models, which usually forces to devote more attention to HMMs. However, we have not followed the structure found in most reviews where HMMs play the central role and other SMs are grouped in a short section mixing model types which, in our humble opinion, are more different between them than to HMM themselves.

The relevance of HMMs also explains why we have dedicated a first section to explain which limitations, usually found in HMMs, have motivated the proposal of SMs *other than* HMMs in the extensive literature. This first analysis has served, for instance, to classify the different sources of variation in a segment realization into extra-segmental and intra-segmental.

implicit vs explicit duration

The main criterion followed here to classify SMs has been the distinction between implicit and explicit duration models. We have shown that this classification is not crystal-clear and that some, so to speak, intermediate categories exist. HMMs constitute the most obviously example of implicit duration models. These models are usually divided into regions. Modeling region sequences is often done by means of finite state devices. To this respect, a review of common HMM topologies for ASR, HTR and similar tasks has been provided.

HMM topologies

Besides model topologies, the other part of SMs is modeling frame sequences. The simplest technique used in HMMs consists in emitting independently thrown frames in a one-by-one basis. Less trivial

39 The reason has been explained several times in this work, in general, and this chapter, in particular.

models describe trajectories (either parametric or non-parametric) or delegate the emission of frame sequences to the evolution of a state dynamics (e.g. by means of recursive, autoregressive or state-space models). Nevertheless, a former distinction prior to this classification, and inspired by [Deng *et al.* 2006], considers first the possibility of generating such trajectoris in an intermediate or hidden level of description. This description is used to obtain the surface or observable frame sequence afterwards. There are several gains in including an intermediate level:

HDMs

- we can talk about trajectories no matter if they are in the intermediate level or in the surface form. For instance, trajectories of parameters driving frame emissions (e.g. parameters of GMMs) allow us to see the generation of independent frames as a particular case which covers constrained mean trajectory models;
- the fact that the intermediate-to-surface level of description includes the identity function makes SMs other than HDMs a particular case of them;
- this mapping may also include quite complex functions that can even operate across frame boundaries, explaining why these models are sometimes called super-segmental (although this comes at a high cost in decoding, even the restriction to apply these techniques to N-best rescoring or only to synthesis and not to decoding).

We can conclude that these gains means a quite general way of describing frame sequence emission.

This also shows that the particular case of generating a single frame can be seen not only as a particular case but also as an important tool used in the intermediate-to-surface mapping. An entire section has been devoted to frame emission covering the most common models and, finally, leading to a general scheme of frame emission types summarized in Figure98. Relating the recent work on this subject, it is mainly related to deep learning techniques that, although they have not been reviewed, we have explained that they perfectly fits the existing proposed scheme. In any case, we can mention that among the advantages of these models and related learning algorithms, there is the nearly systematic improvement of results w.r.t. their previous counterparts, the possibility of using unsupervised data to perform a pre-training and the capability of dealing with raw features avoiding, in some cases, handcrafted preprocessing and feature extraction techniques.

frame emission

The case of context dependent units has been briefly addressed where, besides giving a short glimpse of the state-of-the-art on this subject, we discuss the possibility of halfway-units that, to the best of our knowledge, have not been described elsewhere.

context dependency

Although the use of discriminative models have been relegated to the computation of frame posteriors or to their use in segment-based observations (whose inadequacy in TSGMs has been discussed several times), we describe the way we think they could be used in more general SMs: by using them in combination with generative SMs. Although a short experimentation is provided in Section 15.5 for a HTR task, we believe they deserve a more thorough study in the near future.

discriminative

towards a general scheme

The chapters ends with a little scheme summarizing the previous contents where we basically acknowledge that generative SMs described up to this point do not suffice to cover the space of design (the reason being, basically, the general use of sequence models, the combination of discriminative with generative models and the model combination, at various levels (up to segment level), of both generative and discriminative models).

We would like to conclude this chapter by remarking that, despite of the fact that SMs other than HMMs are usually presented as *the* fair alternative to HMMs, we cannot ignore that HMMs work well in practice, specially when used with frame estimation techniques based on deep ANNs and taking augmented features (using a wide context neighborhood, even the entire sentence if using bidirectional RNNs). Facing the expected remark *"they work well but are inconsistent"*, it is convenient to remember an alternative point of view observed in [Bridle 2004] which consists in viewing HMMs with dynamic features as unnormalized generative models. In any case, we believe that SMs other than HMMs cannot be simply neglected so that, hopefully, many of the models and algorithms proposed in this work, which basically become a superset of most classical ones when departing from HMMs, will be much more justified.

Part II

Proposed Algorithmic Solutions

8 ARCHITECTURE

What I cannot create,
I do not understand.

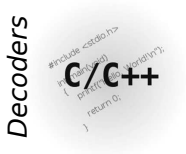
Richard Feynman

CONTENTS

Flexiblity is on of the main desirable properties and an explicit aim of the "'successful trilogy"' which have inspired this work. We have tried to face this feature in at least three different ways:

- the use of programming abstractions such as types, class polymorphism and templates, which allows us to freely replace some data structures and algorithms by others provided that they have the same interface. For instance, different decoders manipulating the same token passing data structure can be used in the same place. These issues are addressed in Chapters 10, 9 and 12;
- a toolkit to experiment with speech, handwriting and other recognition problems is a quite complex system. Different approaches to the high level architecture of these systems can be considered. Some toolkits such as HTK [Young *et al.* 1993], Kaldi [Povey *et al.* 2011] or Bavieca [Bolanos 2012] provide a set of command-line tools following what some people call Unix approach. The user usually makes lots of shell scripts to perform experiments manipulating files. A different approach consists in splitting the system into two parts: kernel and configuration, also known as Two-tiered architecture.[1] The kernel implements the basic

1 See also `http://en.wikipedia.org/wiki/Ousterhout%27s_dichotomy`.



classes, objects and algorithms of the system, whereas the configuration connects these objects to give the desired final shape to the application. The kernel part is usually written in a compiled, statically typed language, like C++. The configuration part is usually written in an extension language [Ierusalimschy *et al.* 1996] which is usually an interpreted, flexible language designed specifically to be embedded into the application. This language replaces shell script languages (e.g. Bash, awk,...) from the Unix like approach. A third option consists on using directly a high level language specially crafted for these applications, as advocated in [Bezanson *et al.* 2012].[2] We have opted for the two-tiered or kernel/configuration design for our system[3] "`April`" based on C++ and Lua. Many independently developed toolkits have also adopted this approach either with other languages (like Tcl/Tk [Soltau *et al.* 2001] or Python [Soltau *et al.* 2010]) or even also with Lua, as is the case of Torch7 [Collobert *et al.* 2011];

- some design requirements are harder than others. For instance, it is convenient, in online speech recognition systems, to start the recognition process at the same time the signal is being produced. This can be achieved by means of on demand computation, where different stages are usually applied one after the other, some ones requiring some delay. A system which can deal with online recognition can also handle the general case. A common way to implement on this feature consists in using pipeline architectures where every module receives, at the input, the output of the preceding one. The pipeline architecture is usually related to de-coupled designs since every stage is sequentially applied. In this work, we propose to follow an architecture known as 'dataflow architecture' [Kahn 1974; Lee and Parks 1995] which is more general than pipelines and which allows the feedback between different components rendering the system more similar coupled architectures, while exhibiting at the same time a great flexibility. Indeed, many different systems can be constructed by combining a little repertoire of kind of 'lego' pieces.

8.1 DATAFLOW ARCHITECTURE

A dataflow network is a network of concurrently executing processes which are connected by directed arcs named channels. These processes, also known as dataflow boxes or components, communicate with others by sending data messages (called tokens, which are typed and may follow a protocol) through their output channels that can buffer those messages. The sender of a message need not wait for

2 Unfortunately, I have not found any single language which I really *like*, not to mention one suitable for this tasks. Julia did not exists when the development of this toolkit started around 2005.

3 Why to construct a new toolkit when there are so many ones "out there"? Many current toolkits where not available where this project started, but the main reason was to have more flexibility at our disposal, namely: acoustic models other than HMM/GMMs, language models other than count-based n-grams and a lot of decoder variations.

the receiver to be ready to receive it. This style of communication is often called asynchronous message passing [Kahn 1974].

loosely coupled architecture

Dataflows are loosely coupled software architectures relatively easy to parallelize and to distribute. They have already proven their convenience for modeling signal processing systems such as the CLAM system [Amatriain 2007] or some well known software toolkits such as Simulink [Dabney and Harman 1997] or Labview [Johnson 1997], among others [Abram and Treinish 1995; François and Medioni 2001]. They have been used recently for describing deep learning models [Abadi *et al.* 2015]. We can also find several systems influenced by this dataflow architecture in the speech recognition field: for instance, the ATK wrapper for HTK [Young 2007] or also the preprocessing step of the RWTH speech recognition system[4] [Rybach *et al.* 2009].

A dataflow box which has only output ports is usually named a source. Analogously, a dataflow box which has only input ports is known as dataflow sink. Sources may represent input devices, whereas a sink may send messages outside the dataflow.[5] A dataflow with an input channel and an output channel is usually called a filter, but more complex dataflow components, from the connections point of view, are sometimes required.

An example case of the dataflow architecture is the use of pipes in the Unix operating system, although it is limited to linear flows of data: a source followed by a chain of filters finally connected to a sink. More complex dataflow architectures are possible, even with cycles. When no cycles are allowed, they can be easily implemented by means of what are known as pull and push architectures. In push architectures the sources send tokens which cause other pieces to react and new tokens are sent. In pull architectures, the sink components demand new data which causes other components to also demand more data until sources are reached.

We have implemented a dataflow architecture which allows cycles and where dataflow channels can buffer several tokens.[6] The most common way to implement this architectures is by means of threads, making it possible the parallelization of different parts of decoders in order to deal more easily with input/output and to take profit of multicore CPUs. Indeed, other systems using this idea can be found in the literature without any explicit mention to dataflow architecture or the associated nomenclature, as is the case of [Fink *et al.* 1998]:

> *separate modules for signal recording, feature extraction, codebook evaluation, calculation of emission probabilities, acoustic beam search, and LM based hypothesis search. All of these are implemented as threads running within the common address space of a single process and use ring buffers for communication.*

4 The dataflow architecture is limited to the preprocessing stage.

5 Other useful examples of sources and sinks are serializer and deserializers of tokens, which can send and receive messages over a computer network, allowing the dataflow to be distributed over several computers without effort. It is even better to make this serialization transparent to the programmer by performing this serialization (or marshaling) automatically.

6 In a recent re-implementation, mainly made by Francisco Zamora, of the connectionist part of our toolkit, the flow graph used in stochastic gradient methods is implemented using dataflow ideas and reusing some code (tokens,...) but using a more simpler strategy since, in feedforward models, there are no cycles.

This example, which is not unique, also shows that these ideas constitute a natural choice when faced to similar problems, but we believe that it is advantageous to provide an explicit implementation of dataflow architecture which, in our case, has been implemented in C++ but is accessible from Lua scripts making it possible to combine the different pieces in several configurations to create entire decoders with scripts.

In order to develop this architecture, we have to specify types of tokens (which may be conceived to follow a predefined protocol) and a repertoire of dataflow components which are described Section 8.3 and which may be implemented as C++ classes using inheritance to capture common features.[7] Although dataflow boxes represent functions mapping input sequences into output sequences,[8] a common way to implement its behavior consists in considering the existence of an internal state. This allows us to express how this state is updated and which tokens are sent in terms of an input token and the current state.[9] These specifications are quite easy to implement when each dataflow box is running on a different thread, although other approaches (e.g. based on co-routines) are possible. For the sake of portability, our first dataflow implementation did not require threads and emulated co-routines, but it was extended later to threads in order to take profit of parallelism. Indeed, a mix between both approaches by means of a thread pool manipulating co-routines and tuned to the proper number of CPU cores seems a good design choice quite easy to implement in our system, probably based on a library such as `libuv` to provide asynchronous event notification.

8.2 DAG SERIALIZATION PROTOCOL

The items or tokens which are sent through the dataflow channels are typed. For instance, a parametrizer can be modeled as a source which processes the raw signal and sends tokens representing frames by means of fixed dimension floating point vectors, but other different types of tokens can be found in other parts of the system. Moreover, tokens are sent following a particular protocol.

One of the data types more relevant in this work are DAGs, which are produced by some boxes and consumed or manipulated by others. DAGs are not sent in a single token. Instead, a protocol has been developed to serialize DAGs. The purpose of this protocol is to allow the decoding algorithms *to perform operations in response to the messages received while the DAG is being transmitted*. They do not need to wait to the end of the description to start. Besides, it is not even necessary to store the entire DAG in most decoding algorithms: only a dictionary of active states associated to the "current vertex" is required is most cases, as will be described in Section 12.2.

7 For instance, we have described a generic filter component so that only the generation of an output token given an input token has to be rewritten to create new filter types.

8 Note that these functions are not defined over individual tokens, but over entire token sequences.

9 Some dataflow components also need to take into account external data or to generate external world actions.

incidence protocol

The proposed protocol is named "incidence protocol" because all edges incident to a given vertex are sent together. This protocol is not a format description or a file representation whatsoever but the specification of a set of message types and the way they should be used. The serialization of these messages is not even required but could be done in several ways.[10]

The use of this protocol is a trade-off between the ease of implementation of DAG manipulation algorithms and the possibilities for parallelization. On the one side, it makes easier the generation, manipulation and analysis of DAGs in terms of simple reactive methods described by means of non-parallel algorithms but, on the other side, algorithms described in this way have an inherently serial description which prevents us to completely exploit the possibilities of parallelism offered by this task.

It is also possible to define a symmetrical "adjacency protocol" where all edges outgoing from every vertex are sent together, but we believe this other protocol is not as useful to define DAG parsers.

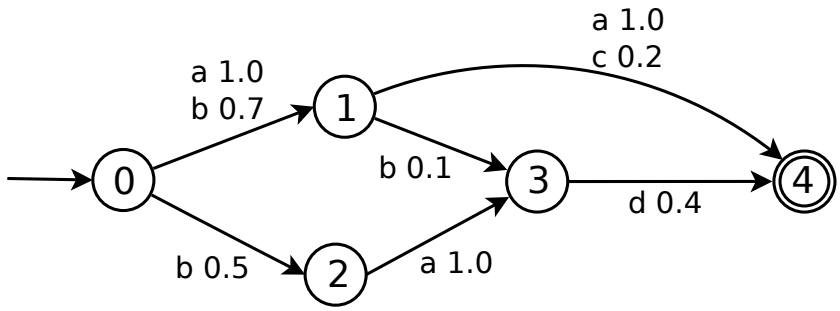
The DAG to be serialized has to be topologically sorted because all edges which share the same destination vertex must be send together and all vertices which are origin of these edges must have already been sent. The following repertory of message types is required:

> `begin_dag`, `vertex`, `edge`, `no_more_in_edges`, `no_more_out_edges`, and `end_dag`

which are used as follows:

- the beginning and the end of the word graph must be notified with `begin_dag` and `end_dag` messages respectively;
- each vertex is sent with a `vertex` message. Vertices are identified with an integer value. From now on, *current vertex* will refer to the last vertex which has been sent so far;
- `is_initial` is an optional flag of the `vertex` message used to notify the current vertex is the initial DAG vertex;
- analogously, the current vertex may be qualified as final by including the optional flag `is_final`;
- each edge is sent with an `edge` message. Each edge contains a list of ⟨symbol, score⟩ pairs, so that a sole edge suffices to send all symbols between a given pair of origin and destination vertices. Note that the origin vertex is specified in the message but not the destination vertex, since the incidence protocol assumes that it must be the current vertex;
- it is desirable to have DAG edges associated to null transitions. These edges are not very common, but they can be useful to model some features (in the same way as null transitions are used in WFSTs where DAGs are just a particular case) such as allowing several initial or several final vertices, to model the recognition of a crossing out which do not require a language model transition, etc. This message has a sole argument corresponding to the score. Note that some DAG generation algorithms described

10 For instance, as is done by the serializer and deserializer dataflow boxes which produce a byte stream suitable to be sent through the network or stored in a file.




```
1: begin_dag(multistage=false)
2: vertex(0,is_initial=true)
3: no_more_in_edges(0)
4: vertex(1)
5: edge(0,data={⟨a, 1.0⟩, ⟨b, 0.7⟩})
6: no_more_in_edges(1)
7: vertex(2)
8: edge(0,data={⟨b, 0.5⟩})
9: no_more_in_edges(2)
10: no_more_out_edges(0)
11: vertex(3)
12: edge(1,data={⟨b, 0.1⟩})
13: edge(2,data={⟨a, 1.0⟩})
14: no_more_in_edges(3)
15: no_more_out_edges(2)
16: vertex(4,is_final=true)
17: edge(1,data={⟨a, 1.0⟩, ⟨c, 0.2⟩})
18: no_more_out_edges(1)
19: edge(3,data={⟨d, 0.4⟩})
20: no_more_out_edges(3)
21: no_more_in_edges(4)
22: no_more_out_edges(4)
23: end_dag()
```

Figure 102: A directed acyclic graph (up) and a message sequence generated to serialize this graph using our incidence protocol (down).

in Chapter 11 can be customized by a control automaton which allows us to control the generation of this type of DAG edge. At least, two different choices are possible to implement null transitions, namely: the use of a special `null_edge` token message, or the inclusion of this score in the `edge` token. In this last case, the null edge can be identified with a special word identifier or as an optional flag;

- the `no_more_in_edges` message is sent after all edges arriving at current vertex have already been sent and before any new `vertex` message is sent. This message does not require arguments. The reception of this message is used to trigger operations which require the previous reception of all incoming edges and it is used by some decoding algorithms: each previous edge message modifies an internal data structure which is then used when receiving the `no_more_in_edges` token;
- `no_more_out_edges` is sent when a vertex which has been already processed has no more outgoing edges, and is used to know when some resources associated to the corresponding vertex can be released (e.g. garbage collection). The only argument of this message is the vertex identifier.

In order to clarify this protocol, Figure 102 shows an example of a DAG and a possible set of messages required to transmit it using the incidence protocol. It could be possible to group, in a sole message, some common dataflow message sequences which are typically sent together. For instance, by grouping in a sole message all the relevant information related to a given vertex:

- a final and final optional attributes;
- the set of all incoming vertices, for each one the set of edges and a boolean value indicating whether or not this vertex has edges outgoing to future vertices.

We believe that the current more fine-grained protocol is more flexible. An adapter between both protocol choices can be easily implemented.

Extension to multi-stage DAGs

Multi-stage DAGs are a particular subclass of DAGs where vertices can be grouped into stages so that incoming edges must come from the previous stage. This particular type of DAG is of interest because some dataflow components are able to profit this feature to improve performance and to simplify the implementation.[11] The proposed extension of the protocol to deal with multi-stage DAGs is as follows:

- the `begin_dag` message includes a parameter to specify whether or not the DAG is multistage;
- `change_stage` token is used to notify the previous stage is closed and a new stage begins;
- `edge` messages are not allowed to come from vertices which do not belong to the previous stage.

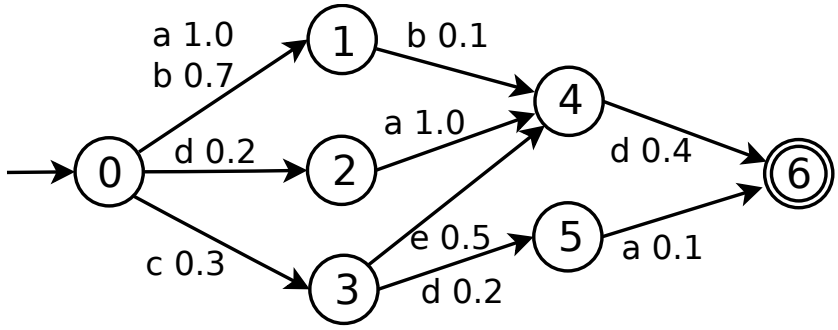
An example of this extension is illustrated in Figure 103.

Extension to acyclic RTNs (packed shared forests)

Packed shared forests (which we consider roughly equivalent to acyclic RTNs) were covered in Chapter 5 and can be used to represent the output of the system when the input sequences are represented by means of acyclic RTNs or when the language models are RTNs.

Let's now see *how easily* the incidence protocol can be extended to deal with acyclic RTNs. The basic idea is to allow the serializer to allow edge messages to reference not only symbols only symbols but also references to previous spans from the DAGs. These spans could be defined by pairs of vertices although it should be desirable to include in those vertices information about their future use in order to easy the implementation of some dataflow components manipulating these structures. Unfortunately, this extension has not been implemented for the moment.

11 This extension has been used, for the moment, in [Zamora-Martínez 2012] where the author has extended the use of the proposed dataflow architecture from sequence recognition to statistical machine translation.



1: `begin_dag(multistage=true)`
2: `vertex(0,is_initial=true)`
3: `no_more_in_edges(0)`
4: `change_stage`
5: `vertex(1)`
6: `edge(0,data={⟨a, 1.0⟩, ⟨b, 0.7⟩})`
7: `no_more_in_edges(1)`
8: `vertex(2)`
9: `edge(0,data={⟨d, 0.2⟩})`
10: `no_more_in_edges(2)`
11: `vertex(3)`
12: `edge(0,data={⟨c, 0.3⟩})`
13: `no_more_in_edges(3)`
14: `no_more_out_edges(0)`
15: `change_stage`
16: `vertex(4)`
17: `edge(1,data={⟨b, 0.1⟩})`
18: `no_more_out_edges(1)`
19: `edge(2,data={⟨a, 1.0⟩})`
20: `no_more_out_edges(2)`
21: `edge(3,data={⟨e, 0.5⟩})`
22: `no_more_in_edges(4)`
23: `vertex(5)`
24: `edge(3,data={⟨d, 0.2⟩})`
25: `no_more_out_edges(3)`
26: `edge(3,data={⟨d, 0.4⟩})`
27: `no_more_out_edges(3)`
28: `no_more_in_edges(4)`
29: `change_stage`
30: `vertex(6,is_final=true)`
31: `edge(4,data={⟨d, 0.4⟩})`
32: `no_more_out_edges(4)`
33: `edge(5,data={⟨a, 0.1⟩})`
34: `no_more_out_edges(5)`
35: `no_more_out_edges(4)`
36: `end_dag()`

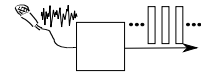
Figure 103: A multistage DAG (up) and a message sequence generated to serialize this graph using the extension of the incidence protocol to deal with multistage DAGs (down).

8.3 OVERVIEW OF DATAFLOW COMPONENTS

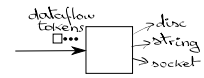
This section briefly describes some dataflow components which will be covered in more detail in next Chapters. They are described here because they are needed in the following section to describe some recognition systems in terms of dataflow networks which make use of these components. Their description is quite brief and no implementation details are provided.[12]

8.3.1 Generic dataflow components

Let us describe some dataflow components which are not specially related to our tasks:



- **dataflow source** is the generic name of box with no inputs but some outputs. A typical dataflow source may represent an input device (e.g. a microphone), a file, a set of patterns, etc.
- **sink** is the generic name of any dataflow box with inputs but no outputs. From the dataflow perspective, a sink component simply drops all incoming tokens but, in practice, incoming tokens may be stored into a file and so on;



- **serializer** is a special type of dataflow sink which converts any incoming token to a serialized byte stream which can used outside the dataflow (for instance, to send it through network or to store it into a file). This component delegates on a method of the token class that each different type of token has to overwrite and which places an identifier of the type of token as the first value of the byte sequence. The combination of serializer and deserializer allows us to split a dataflow component into several computers making very easy the development of distributed recognition systems;
- **deserializer** is a dataflow source which receives a byte stream, corresponding to serialized dataflow tokens, and converts them back to a token sequence which are emitted to the dataflow. The byte stream comes from outside the dataflow network and, therefore, it is not visible in terms of dataflow. This component is also based on the use of a homonym (static) method defined in the token class hierarchy. Since the type of the token is identified with the first byte of the token description, it is possible to know to which particular class of token class we have to delegate;

- **filter** is the generic name of any dataflow which has one input and one output and which calls a function every time a token is received in order to send the function response token by the output channel. The function performs some type of transformation to the incoming token. Many different filters sharing this basic structure can be easily implemented by means of inheritance where only the function has to be redefined;
- **spy** is a particular case of the previous filter using the identity function. That is, a filter which sends the tokens without modify-

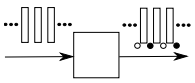
12 Some implementation details are, nevertheless, explained in the introduction of Chapter 11.

ing them. This dataflow component does not perform any useful work from the dataflow point of view but it can be very useful for debugging purposes (visualize information relating the tokens);

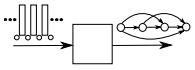
- **switch** allow us to connect any number of input channels and output channels. Any token received by one input channel is send by one output channel. It is possible to configure the operational mode to specify in which order the input and output channels are used (for instance, round robin) or if tokens should be replicated.

8.3.2 Domain specific components

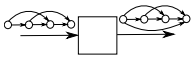
Likewise, it is also possible to enumerate some domain specific components for our recognition tasks at hand:



- an **external over-segmenter** receive a sequence of frame feature vectors and generate the same sequence with some interleaved tokens relating to the classification of over-segmentation frame boundaries. As we can expect, this module, like the parametrization itself, is domain specific, although some general algorithms can be envisaged.[13] The use of the produced over-segmentation information is discussed in Sections 11.1.4 and 12.4;



- **seqgraphgen** receives a sequence of vectors representing frames interleaved with tokens with oversegmentation information (the output of the external oversegmenter just described) and produce as output a "input word graph" DAG using the DAG incidence protocol described in Section 8.2. Note that edges may be labeled with lexical units[14] other than words: it can generate DAGs of phones or DAGs of graphemes. This module is explained in detail in Section 11.1, where we will explain how it can be used with context dependent units or how the use of feedback information can be used to accurately prune the search space in order to improve its performance;

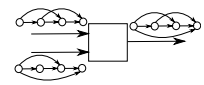


- **graphgraphgen** is a sort of generalization of the previous dataflow component, since a sequence is a particular case of a DAG and the proposed module receives a DAG instead of a sequence. Indeed, both models can be considered as a particular specialization of transducer composition *specially tailored to work on the fly with DAGs serialized with the incidence protocol*. It can be used, for instance, to obtain a word graph from a phone/grapheme graph. That it, edges of the output graph corresponds to paths or spans of the input graph using a lexicon to filter which ones are valid and taking into account only those spans which are reachable by previous ones. This, as described below, corresponds to a particular case of transducer composition which can also take into account edition operations (insertions, substitutions, deletions). This dataflow component is described in Section 11.2;

13 Some over-segmenters, such as those described in Section 14.7 are not based on frames, so they do not belong to this category.

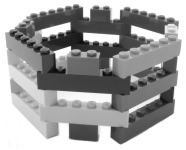
14 See Section 6.3.

- **DAG decoder** is the module which receives an input word graph and combine it with a language model in order to obtain, as output, a trellis which implicitly represents the set of best sequences (in the Viterbi sense). This output can be used to obtain the best sequence, the n-best ones or an output word graph (in some way, a reduced trellis). This component corresponds to the second stage of the two stage decoder and is studied in Section 12.2. An interesting application of this decoder consists of reducing or pruning *the input graph* by marking which parts of this input have finally contributed to the surviving hypothesis of the decoding process. This reduced graph can be used later by a more sophisticated (and possibly more expensive) decoder, as discussed in Section 8.4.4. Note that this pruning could be implemented in a more efficient way than other multi-pass approaches where the LM information is removed from the word graph, obtained by the first decoding step, which is reduced afterwards;
- **one-pass-decoder**, is a dataflow module which receives frames interleaved with oversegmentation information as with the previous `seq_graph_gen` component. Instead of constructing an input word graph, it directly computes the Viterbi trellis associated to the one-pass approximation. This module basically performs a time-synchronous Viterbi decoding which is more efficient than the combination of a `seq_graph_gen` and a DAG decoder. This module is studied in Section 12.3 where several issues, such as the use of oversegmentation information or the use of several threads to take profit of multi-core architectures, are discussed;
- **DAG combination** is specially useful in the case of bimodal (or multimodal) recognition. Multimodal recognition is not very difficult when we try to perform sequence classification (see Section 3.1.1), as is the case of isolated word recognition (see Section 15.6 for some experimental works on this subject). When the sequence to be classified may be composed of several words, the problem is similar to that of multi-tape/multi-stream recognition where the segments from each tape can be aligned in many ways with segments of the others. This can be easily solved at the first stage of the two stage recognizer by combining the input DAGs associated to every modality. The proposed module simply performs this combination (some kind of cartesian product). As with other components receiving DAGs, it is adapted to receive each input DAG as well as emitting the output DAG using the incidence serialization One of the main advantages of this approach is the possibility of using a standard DAG decoder. This dataflow component is described in Section 11.3;



- **DAG filtering** will be used in this work to refer any dataflow component which receives and output a serialized DAG and which performs a filtering at the DAG level. For instance, an input DAG can be filtered to take into account information provided by the user of an interactive transcription tool. For instance, if the user has already validated a prefix of the sequence, the filter can prune all parts of the input graph which do not correspond to that prefix.

8.4 RECOGNIZER EXAMPLES



For the moment, we have only mentioned some types of "lego pieces" that we have at our disposal. This section describes some examples of recognizers using dataflow networks in order to situate this pieces in context to better understand their purpose, to show how these pieces can be combined together in different ways and to illustrate the flexibility of the proposed architecture.

8.4.1 One pass

The so called one-stage recognizer (see Section 12.3) is one of the most wide-spread models reported in the literature. The segment models (e.g. acoustic models in ASR) are HMMs and the language models are also regular models (usually n-grams), so their composition is also regular. The resulting search space is a trellis which can be processed time synchronously, one frame at a time, or also by means of other techniques such as stack decoding. This leads to a simple and linear[15] dataflow architecture whose most relevant module receives frames at the input in order to construct the trellis, as shown in Figure 104. We can observe in this figure, from left to right: a source (representing a microphone in the case of ASR), a parametrizer,[16] an optional external oversegmenter (which makes more sense in the case of HTR than in ASR) and the one-stage decoder.

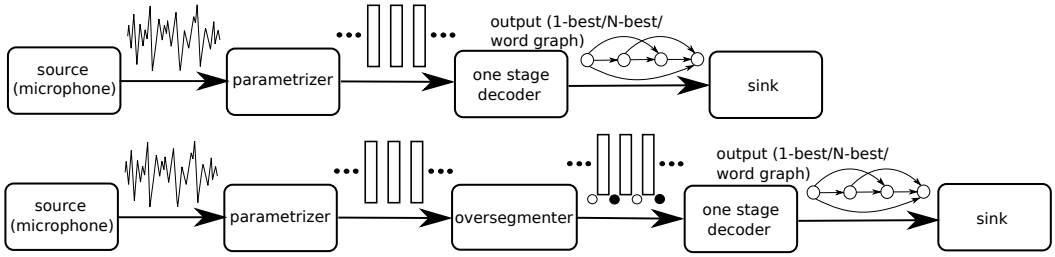


Figure 104: Dataflow architecture of a one-stage recognizer without oversegmenter (up) and with oversegmenter (bottom).

The use of oversegmentation information in this type of decoder is perfectly possible, as will be described in Chapter 12. Surprisingly, to the best of our knowledge, this possibility has not previously appeared in the literature. Relating the output, the word graph output could also be emitted using the incidence protocol.

8.4.2 Decoupled and semi-decoupled architectures

pejorative connotations

The term "decoupled" traditionally refers to a cascaded approach where a first decoder outputs a sole hypothesis (e.g. a sequence of basic labels) which is converted into a word sequence afterwards. This approach is not robust at all since a mistake in a stage is very difficult

15 Indeed, it can be implemented using a Unix style pipeline approach.

16 It is also possible to integrate the parametrization and a voice activity detection (VAD) in the source component.

or impossible to recover in the following stages. That is why this term has had very pejorative connotations, which explains why we prefer to use the term "semi-decoupled" to avoid any type of confusion with the much more limited approaches situated in one extreme of a continuum. Indeed, a decoupled architecture generating a sufficiently large set of hypothesis at each stage could obtain, in principle, the same results than an integrated decoder *excepting the differences caused by pruning their respectives search spaces*.

(semi-)decoupled vs multipass

It is also quite easy to confuse a semi-decoupled and multipass decoders and, indeed, the nomenclature overload usually found in the literature does not help very much. Roughly speaking, the first stage of a semi-decoupled architecture does not take into account the lexicon and the language model, which are used by the second stage. In a multipass decoder, the first stage is a complete decoder using lexicon and language model wheras the second stage purpose is to rerank the set of hypothesis from the first stage by means of more complex and possibly more computationally costly models. Multipass architectures are described in more detail in Section 8.4.4.

Dataflows can trivially be used to implement these pipelined architectures where each stage of the two stage generative model (see Section 4.2) is performed by a different component, as illustrated in Figure 105, where a module (called `seq_graph_gen` and described in Section 11.1) produces an input DAG (i.e. where each edge is labeled with the likelihood associated to lexical unit). This input DAG is combined with a LM in a "DAG decoder" (described in Chapter 12) which produces an output wordgraph or, alternativelly, a set of N-best sequences.

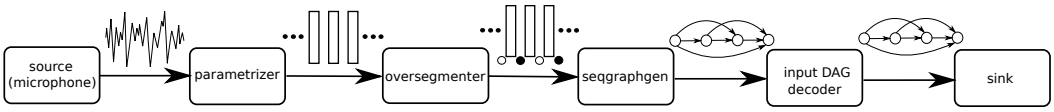


Figure 105: Dataflow architecture of a semi-decoupled recognizer.

Figure 106 illustrates an alternative way to obtain the input DAG (usually labeled with words). It consists of first generating an input DAG of basic units (i.e. phonemes, graphemes,...) which is converted into another input DAG of words by means of a module called `graph_graph_gen` (described in Section 11.2) which uses a lexicon but not a language model. The rest of the dataflow remains unchanged. Note also that it is possible to produce an input DAG of basic units and to use it directly by the DAG decoder without its prior conversion into an input DAG of words. In this case, the DAG decoder would use a LM where lexical units are those basic subword units (described in Section 6.3.2 and known as "lexicon free approach").

Although semi-decoupled architectures seem less efficient in the most common case of HMM and n-grams due to the quadratic[17] cost of the input DAG obtained, they also exhibit some merits which are

17 At least quadratic in space when producing a complete DAG. Oversegmentation can reduce this. Pruning leads to an average segment length (an upper bound could be forced, anyway) which leads to a linear cost. The temporal cost depend on the capability of segment estimators to reuse information from overlapping segments.

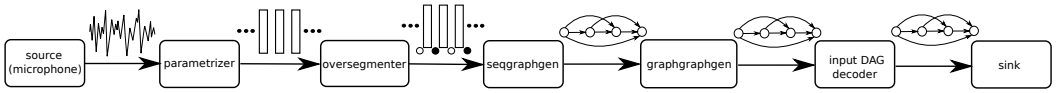


Figure 106: Dataflow architecture of a decoupled recognizer where a `seq_graph_gen` produces a DAG labeled with basic or sub-word units that is later converted into a DAG of lexical units to be finally combined with a LM in the DAG decoder to produces the output.

increase of flexibility

roughly summarized as an *increase of flexibility* in many aspects in general and with respect the type of segmental models, which can be used in the generation of label and word emission likelihoods, in particular. Segment models other than HMM are, in general, not compatible with the one pass approximation. Let us see two additional examples of this flexibility: on the one side, as illustrated in Figure 107, we can obtain several input word graphs using different models which can be combined afterwards (using a module described in Section 11.3). On the other side, decoupled architectures has some advantages, even when they are based on HMM and n-grams, if the same input signal has to be analized with several LMs since, as shown in Figure 108, several DAG decoders take profit of the input graph generator effort. An example of an observed signal which has to be analized with several LMs can be found in the field of automatic dialogue systems where the dialogue manager might maintain several hypothesis and to use LMs specially adapted for each one.

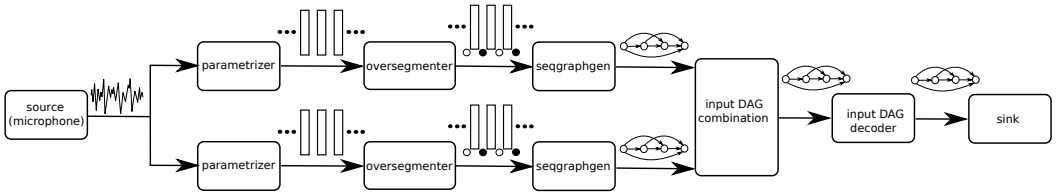


Figure 107: Dataflow architecture of a cascade or decoupled recognizer with several input DAG generators which are combined before being sent to the DAG decoder.

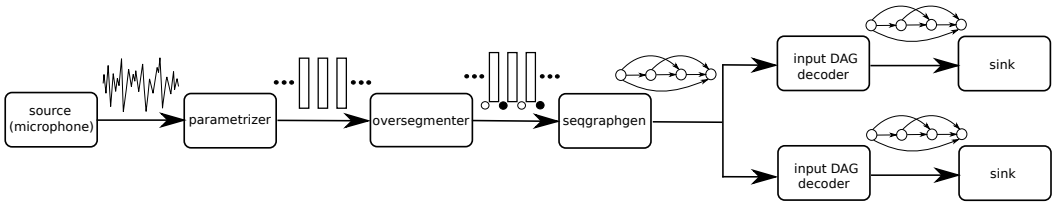


Figure 108: Dataflow architecture of a decoupled recognizer with several DAG decoders sharing the same input DAG.

8.4.3 Semi-decoupled with feedback

In the previous architecture, the generation of the input word graph (with or wihout the phone generation intermediate step) produces a set of hypothesis which must be pruned for practical reasons. Pruning is risky since there is no way to recover when required hypothesis are lost. In a coupled architecture there is also a pruning, but both the acoustic and the language model sources of information are considered *jointly*. So pruning is what contributes the most to make different coupled and cascade architectures.

Being said that, a clear drawback of the decoupled approach is the use of partial information at each stage instead of taking all the available information at the same time. *One of the aims of our work is to show that some decoupled architectures can take into account the same type of global information as the one pass counterpart*, at least relating the pruning, and that they can even be implemented in a flexible way by means of dataflow architectures. We will show how to add this capability of joint LM and acoustic information to pruning to the proposed dataflow architecture while preserving, at the same time, the orthogonality so that several graph generation and graph analysis components can be combined in a "plug-and-play" way. The idea consists of adding a new dataflow channel which sends tokens in the backwards direction, from the DAG analysis module to the graph generation one, as shown in Figure 109. Whenever the graph generation sends a `no_more_in_edges` message associated to the current vertex, the graph analysis module sends by the feedback channel the best combined score (LM score and input graph score) of the trellis nodes associated to the current vertex just sent. This information can help the graph generator to prune *with the same information as used in a coupled recognizer* as described in Section 11.1.3 and in Chapter 12. We have also proposed the inclusion of information from the language model into the feedback message which will allow the input DAG generator module to choose the most specific lexicon in each case.

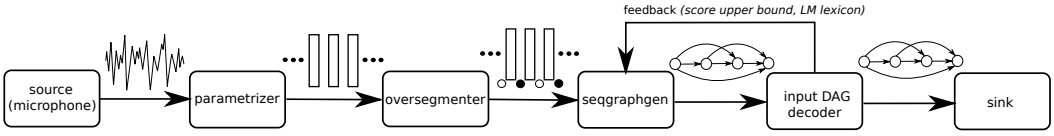


Figure 109: Dataflow architecture of a recognizer with a feedback between the graph analysis and the graph generation modules.

8.4.4 Multipass recognizer

The main purpose of multi-stage recognition is to make practical the use of complex and expensive acoustic or language models by applying them to a reduced search space previously obtained with cheaper models. For example: the first stage may use a cheaper bigram LM to obtain a reduced set of hypothesis. Later, a higher order more expensive connectionist language model can be applied over this word

graph. At least two alternative techniques can be used to implement this multipass approach (illustrated in Figure 110):

- the output of the first decoder is used as the input of the following one, in this case the scores must distinguish which part is due to the likelihood of observing the segments, given the hypothesized unit, and which part is the contribution of the LM, since this last contribution has to be removed when combined in the following pass. Note that even if we remove the LM contribution, there may be several hypothesis associated to the same labels and they could be combined by techniques as on the fly determinization of WFSTs;

- the first decoder is used to prune the input DAG before sending it to the second decoder. In this case, the scores associated to edges remain unchanged but those edges which have not contributed to the best surviving hypothesis in the first decoder are removed. We have not found any prior description of this technique in the literature although, as we have already pointed out, this will probably be more a question of the difficulty to find things in the prior art since it seems to us highly unlikely that nobody had tried that before.

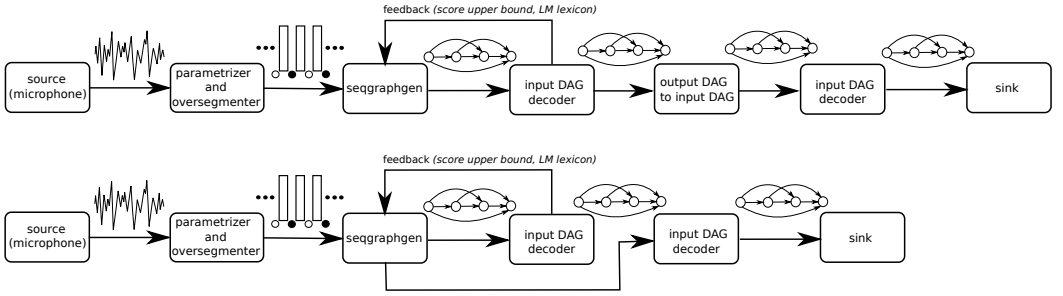


Figure 110: Dataflow architecture of a multipass recognizers where: (up) the second pass uses the output of the first pass where LM scores are removed and, optionally, the DAG is reduced and (bottom) the first pass is used to prune the input DAG before sending it to the second pass.

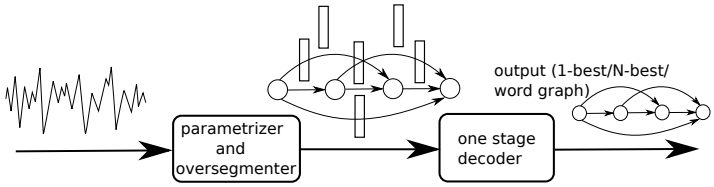


Figure 111: Dataflow architecture for a one-stage dag input recognizer which is useful in cases where preprocessing ambiguities are delayed to later stages where more information is available.

8.4.5 One-stage dag recognizer

dag of frames

Nothing prevents us from extending the one pass decoder from frame sequences (with optional oversegmentation tokens interleaved with frames) to the more general case of a DAG of frames, as shown in Figure 111. Similarly, the `seq_graph_gen` module can receive a DAG of frames to produce either a DAG of words or a DAG of subword units which could be used later the the same `graph_graph_gen` without any problem. The extension of the one pass decoder from sequences to DAG of frames will be described in Chapter 12 and is useful to delay preprocessing ambiguities to later stages where more information is available as explained in Section 2.9.3.

8.4.6 Multimodal recognizer

As explained in Section 3.2.1, the problem of multimodality, from a decoding point of view, is more related to the existence of several input sources which we know are associated the same underlying hidden sequence that to the true existence of several modalities as this concept is understood in the field of human-computer interaction.

An example of multimodal recognition occurs when someone has read a sentence and we have the scanned document and the recorded audio at our disposal. A more realistic example would be the case of a multimodal interactive transcription system but this example, as it is related to interactivity, will be delayed to the next section.

It is clear that, in those examples, the observed sequences cannot be aligned at the phone and at the grapheme level respectively because the transcription of a word into phones is not performed in a grapheme by grapheme basis and since phone and grapheme context dependencies may be different.

This is the reason why, depending on the different input sources, we have to choose at which level the information sources should be merged. In this case, we believe that alignment at the word level would be a good choice, as depicted in Figure 112 where input word DAGs are computed separately for each type of modality. Both graphs are combined on the fly (see Section 11.3) and both can take advantage of the feedback information from the DAG decoder.

Note that, although it is also possible to adapt one-stage recognizers to deal with multimodal input, the advantage of the proposed architecture is that all the components have been previously designed for other type of recognizer so that only the configuration of the dataflow graph has to be adapted, which is an example of flexibility.

8.4.7 Interactive assisted recognition

In the interactive assisted paradigm, described in Section 3.2.2, an "oracle" or user interactively validates and corrects the transcription proposed by the recognizer. The system enters into a loop until the sequence to be recognized has been completely validated and, therefore, the number of steps is not known a priori.

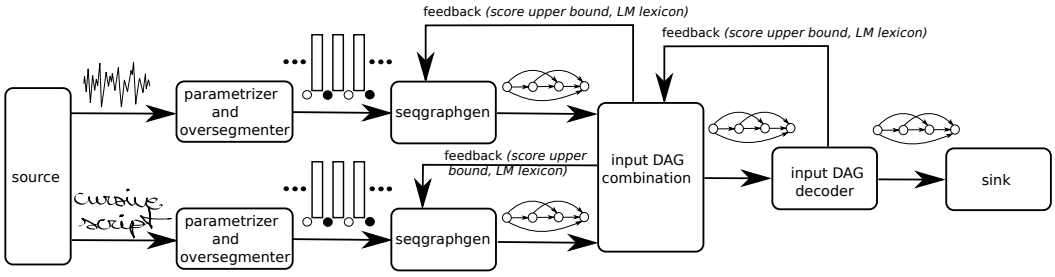


Figure 112: Dataflow architecture of a multi-modal recognizer where we have supposed that a handwritten line and a recorded speech contain the same word sequence. Information fusion is performed at the word level since it makes nonsense to combine at the subword level.

That is why it is better to describe a dataflow architecture suitable *for one of these interactions* which is capable of taking into account the information computed in previous interaction (in order to reuse it) as well as the information provided so far by the user. The overall system would provide a graphical user interface and would use the dataflow recognizer for each interaction. Nearly all previous dataflow architectures proposed before can be adapted to this new scenario, but we have not graphically depicted the dataflow scheme since it would remain essentially unchanged for one reason: We believe that it is better to rely on the capability of dataflow components to store and manage internal data structures rather than to send information among them by means of channels.

Let us check how each component is affected by the inclusion of this feature in the case of a two stage decoder and in an left-to-right interaction protocol where the user validates prefixes:

- the preprocessing is *usually* not modified since this step does not depend on the validated transcription. This has not to be confused with the capability of users to inspect and to correct some preprocessing stages;
- the parametrization, feature extraction and the estimation of emission probabilities (in the particular case of HMMs) can also be reused in following interactions;
- the generation of the input DAG by the `seq_graph_gen` can take into account the validated prefix to prune the incompatible parts of the input DAG as well as to recompute those parts which were not found due to the existence of pruning techniques;
- the same reasoning can be applied to the internal trellis data structure associated to the DAG decoder.

These issues will be addressed into more detail in the respective chapters associated to the generation of input DAGs and to DAG decoding (Chapters 11 and 12, respectively).

8.5 SUMMARY AND SOME CONCLUSIONS

This chapter has described some aspects of our implementation related to the flexibility, namely, the combination of three features:

- the use of programming abstractions in the form of programming interfaces to deal with different language models, lexicon and segment estimators, etc. in a replaceable way;
- a two-tiered architecture where most computationally intensive parts are written in C++ while configuration scripts and less critical parts may be written in the Lua scripting language; and
- a dataflow architecture where a set of general and specific components can be efficiently implemented in C++ but can be used from the Lua configuration scripts to design a wide and diverse variety of decoders.

dataflow recognizers seem novel

Most part of this chapter is devoted to the dataflow architecture. The different APIs used by decoders, lexicon estimators, etc. are described elsewhere. Although dataflow architectures have been previously used in the field of signal processing and even by some ASR toolkits, their use has been usually restricted, to the best of our knowledge, to the signal processing parts. Otherwise stated, its role is limited to some kind of wrapper over a previously existing decoder. Decoders that seem to work in a dataflow way have not designed a general dataflow architecture but contain ad hoc code usually based on the use of internal buffers and threads.

novel DAG serialization protocol

One of the most interesting features proposed in this chapter is the capability of dataflow components to send and to receive DAGs serialized in such a way that it is not only possible to send them in an incremental way but also to send them while they are being produced, to use them while they are being received *and to combine their production and their use by means of feedback channels which equates decoupled models with integrated ones.* This is reflected in the design of decoders and other DAG manipulation components which are, when possible, specially tailored to the DAG serialization protocol and programmed in a reactive way.

large repertoire of motivating examples

The dataflow architecture is useful to describe hand made architectures and could also be extended to automatically check some of their properties which could easy the systematic exploration of alternatives. We have illustrated the interest and flexibility of this architecture with examples but, obviously, the reader is free to devise many more.[18]

18 For instance, as an exercise, how can be constructed an over-segmenter which uses the segmentation obtained by a one-stage decoder? In this way, internal segmentation obtained by a decoder could prune the search space of a more complex one.

9 LM INTERFACES AND REPRESENTATIONS

MI MÁS SENTIDO ARÁCNIDO.[1]
Spiderman

CONTENTS

MODULARITY and interfaces are key ingredients to achieve flexibility. These interfaces can be placed at different levels: at the object oriented level, when defining data protocols for the dataflow architecture described in Chapter 8, and so on. This chapter studies the implementation of some LMs described in Chapter 6 and the interfaces used to communicate with decoders. We have decided to describe them in a different chapter to better divide this writing into parts as has been explained in Chapter 1.[2]

This chapter is mainly divided in two sections. The first one addresses LM interfaces, where some extensions are proposed. The second one describes some LM representations. The main goal is to reduce the memory footprint and the time of LM look-ups. Besides count based n-grams, other types of LM representation are also covered. In particular, NN LMs have been used in many experiments and a new type of NN LM is proposed in Section 9.5.4. A final section summarizes contributions and draws some conclusions.

1 Personal joke to illustrate a limitation of bigrams: Spiderman is offering condolences in Spanish. Most usual Spanish condolence is *"Mi más sentido pésame"* (our deepest sympathies), but one of the most frequent Spiderman bigrams is *"sentido arácnido"* (spider sense), so the most probable word after prefix *"Mi más sentido"* is *arácnido* and not *pésame*.

2 Three parts corresponding to: 1) theory/models, 2) engineering/algorithms, and 3) tasks/know-how/experimentation.

9.1 LANGUAGE MODEL INTERFACE

The use of general interfaces to connect the LM with other parts of recognition systems is not a new idea [Soltau *et al.* 2001]:

> *To run the decoder with arbitrary linguistic knowledge sources as statistical n-grams, context free grammars, or word graphs, we use an abstract interface between the decoder and the linguistic knowledge sources. The interface consists of few functions to manipulate the linguistic state. The decoder itself works independent from the actual linguistic knowledge source.*

Similarly, [Huijbregts 2008; Section 6.1.4]:

> *The decoder only interfaces with the language model through its API when a new prior needs to be calculated. This procedure makes it possible to easily switch language model implementation.*

In this section, two widely used LM interfaces are described, namely:

- an ad hoc interface for n-grams; and
- a more general interface for LMs based on the chain-rule based decomposition of probabilities or automata representations.

Additionally, extensions of the last interface are proposed. In particular, we are interested in an interface for RTNs. Before deepening into details, let us first describe some features orthogonal to all of them.

Object oriented API and LM servers

LM servers

LM interfaces are usually implemented as abstract classes. In this way, different LM implementations can be replaced without effort. Besides this object oriented API, LM interfaces can also be defined as protocols to communicate with LM servers. For instance, [Federmann 2007] describes such a protocol, for an n-gram based interface. The use of LM servers based on inter-process communication makes it possible to share a LM by several processes. But, when they are based on a network protocol, they can also be distributed over multiple computers [Brants *et al.* 2007]. This is specially relevant when LMs are really huge and when several recognition engines, possibly on different machines, share a common LM. Note that the same LM server can support several interfaces.

Vocabulary queries

Some LM APIs and server protocols are not restricted to LM look-up operations but they usually offer the possibility of querying vocabulary words. For example, when words are associated to numerical identifiers, the possibility of converting from strings to numerical representation and vice-versa is usually available.

Some systems, such as the Janus/IBIS ASR [Soltau *et al.* 2001], propose the use of objects to map sequences of words into word phrase identifiers. In particular, it is very easy to map word identifiers generated by the lexicon decoders into classes to obtain class-based or category-based LMs. We believe that the perfect hashing with FSA [Daciuk *et al.* 2005] is a quite promising technique for these purposes.

9.1.1 n-gram interface

The most widely used type of LMs are n-grams and, not surprisingly, a large quantity of implementation alternatives has been proposed in the literature. But, which interface is best suited for these LMs? Although we believe that the best one is the automaton-based interface described below, the n-gram interface is described here for the sake of completeness.

A LM look-up, in the n-gram interface, receives the LM context and the next lexical unit that the LM user (usually the decoder) is hypothesizing. The LM context, in a n-gram model, consists of the last $n-1$ lexical units. These lexical units are represented by means of character strings or by means of numerical identifiers, but can be generalized to objects representing, for instance, factored representations.

An example of this interface is found in SRILM [Stolcke 2002]. SRILM is one of the most widely used toolkits to train count-based n-grams. It offers several command line tools, but it also offers a C++ library where different LM alternatives are implemented as classes inheriting from a generic LM class. In order to define a new LM class, it basically suffices to define the following method:[3]

```
LogP wordProb(VocabString word, const VocabString *context)
```

which computes the conditional log-probability of **`word`** given a history. The history is given in reversed order (most recent word first) and is terminated by a special sentinel value **`Vocab_None`**. In this way, it is quite easy to define new classes for n-gram LMs. Note that a decoder using this LM interface has to represent each LM hypothesis in the form of a $n-1$-gram and, when applying a LM transition, the next LM context has to be computed by scrolling the vector and appending the last word. This is one of the basic differences with the automaton-based interface described below.

9.1.2 Automaton interface

Weighted transducers (WFSTs) can be used to represent the knowledge associated to all the stages of the recognition in a uniform way [Riley *et al.* 1997]. The static network decoding approach takes advantage of that to offer a quite elegant solution for decoding.

Relating the use of this formalism to represent LMs, it is widely known that n-grams are a particular class of regular languages [García and Vidal 1990]. Several authors have proposed to use WFSTs to represent n-gram models [Riccardi *et al.* 1995; Bordel *et al.* 1997; Llorens Piñana 2000; Vidal *et al.* 2005b]. But note that other types of WFST-based LMs exists besides n-grams. For instance, LMs can be estimated from data by means of grammatical inference techniques. It is also possible to generalize automata based n-gram representations by means of grammatical inference ideas such as the use of state-merging [Bonafonte and Mariño 1996].

3 http://www.speech.sri.com/projects/srilm/manpages/LM.3.html

The automaton interface proposed by [Mohri *et al.* 1998] and followed by others [Finke *et al.* 1999; Kanthak and Ney 2004; Lieb 2006] provides the following methods for using weighted automata:

- **`start`** returns the initial state of the automaton;
- **`final(state)`** returns the final state;
- **`arcs(state)`** returns an iterator that allows to enumerate the outgoing edges of the state given as argument. Every arc contains the following fields: input label, output label, weight and destination state.

By deriving this abstract class, it is possible to represent both static automata and automata generated on-the-fly resulting from local operations. This abstraction provides three advantages:

- the internal representation of automata can be hidden;
- the same algorithms can operate on multiple representations;
- it allows on-demand implementations of several operations.

The advantage of on-demand execution is that only the part used during decoding has to be computed. It is possible to implement a given function on demand if we can specify the set of transitions leaving a particular state, in the resulting automaton, by using only local information from the source automata [Mohri *et al.* 1998; Caseiro 2003; Hori *et al.* 2007]. Transducer composition, union, concatenation, Kleene closure and determinization are well known examples of algorithms which have lazy implementations. However, there are other algorithms, such as the standard DFA minimization or shortest paths computation, which do not admit this lazy implementation.

A problem with this interface, for some decoders, is that we have to iterate over the set of outgoing arcs to find the transition required for a particular word emitted by the lexicon decoder. That is probably why other similar interfaces contain a method to query for a particular transition. The interface described in [Soltau *et al.* 2010; section 2.4] is very similar but contains two different variants for the **`score`** method. The first one assumes the use of complete and deterministic models, as is the case of n-gram LMs. The second one retrieves the scores for all words in the vocabulary associated to **`state`**, which is very useful to compute LM look-ahead scores (LMLA was described in detail in Section 10.6.7):

```
class LM:
virtual STATE start (WORD wordX);
virtual STATE extend(STATE state, WORD wordX)=0;
virtual SCORE score (STATE state, WORD wordX)=0;
virtual void  score (STATE state, SCORE *scoreptr)=0;
```

Moses framework [Koehn *et al.* 2007] also defines language model classes which seem to provide, at the same time, both n-gram and finite state automata methods. Worth mentioning is OpenFst library [Allauzen *et al.* 2007] which is being used by several HTR, ASR and SMT decoders such as, respectively, OCRopus [Breuel 2008], Kaldi [Povey *et al.* 2011] or Ncode [Crego *et al.* 2011], among others.

Proposed automaton interface

A LM interface related to the previous automaton based interfaces is proposed in this work.[4] It has been implemented as a C++ template in order to parametrize the representation of states,[5] and scores (usually log-probabilities). Indeed, two different abstract C++ classes are used to distinguish between the LM representation and their actual usage:

LMMODEL contains the "static" representation of the language model. There can be different representation alternatives;

LMINTERFACE usually contains other data structures to perform the actual computation of LM queries.

This separation allows the possibility of using several decoders with the same LM instance: the **LMModel** is meant to be thread-safe and shared among several instances of **LMInterface** running on different threads. Let us remark that these are abstract classes and that the actual implementation can be tailored for different LM types. Regarding the **LMModel** class, its basic methods are:

```
virtual bool isDeterministic() const;
virtual int ngramOrder() const;
virtual bool requireHistoryManager() const;
virtual LMInterface<Key,Score>* getInterface();
```

The first ones are used to query some LM properties. The **ngramOrder** returns -1 for non-ngram LMs. The **requireHistoryManager** manager is described into more detail in Section 9.4 and the **getInterface** method creates a new instance of a **LMInterface** object. This last class contains the most relevant methods for our current exposition:[6]

```
LMModel<Key, Score>* getLMModel();
void get(Key key, WordType word, Burden burden,
       Vector<KeyScoreBurdenTuple> &result,
       Score threshold);
void getNextKeys(Key key,WordType word,vector<Key> &result);
BasicArcIterator basicArcs(Key key,Score threshold);
Key getInitialKey();
bool getZeroKey(Key &k) const;
Score getFinalScore(Key k, Score threshold);
Score getBestProb() const;
Score getBestProb(Key k);
```

The **getLMModel** method can be used to retrieve the underlying **LMModel**. Observe that nothing prevents programmers form inheriting from both abstract classes simultaneously: the **getInterface** and **getLMModel** methods would return the object itself (**this**).[7]

4 This design is a collaboration with Pako Zamora and Adrián Palacios.

5 Nevertheless, this library is not intended to be suited for arbitrary FSA equipped with generic semiring types. Note also that we are using LM context, key, state or LM identifier as synonyms.

6 The **virtual** qualifier has been removed in order to simplify the code listing.

7 Note that, in this case, the object should not be used by several decoders simultaneously since the bunch mode, described below, is not easily implemented in a thread-safe manner.

The most basic method for LM queries is **get**, which receives a LM **key** and a word identifier and produces zero, one or more destination states[8] (LM keys) together with their corresponding transition probabilities. Some details are worth mentioning:

accompanying burden value

- there is a value which may accompany a LM queries and which is basically ignored by the **LMInterface**. This is the reason why it is called **Burden** in this context.[9] The rationale for including this parameter is twofold: On the one side, this makes the use of LMInterface easier for decoders, as explained into more detail in Chapter 12. On the other side, it is used by the LM to simplify the bunch mode LM interface described below;
- the result is returned by storing zero, one or several values in a vector received as parameter. The elements of this vector are from the **KeyScoreBurdenTuple** class, which is basically a tuple consisting of a Key, and Score and a Burden value.
  Observe that this interface differs from previous automaton interfaces which allowed us to iterate over the set of outgoing transitions. Now, we cannot iterate but only query for the outgoing transitions associated to a given label.[10] However, it is possible to have several transitions associated to the same label, which means that non-deterministic models can be represented. We believe that non-deterministic interfaces may have advantages in some cases. For instance, when a nearly deterministic model is used by dynamic decoders[11] without applying determinization;
- a threshold value is used to enable the use of pruning techniques *within the LM*. This extension is described below.

There is a method related to **get** and called **getNextKeys** which only returns the set of LM destination states. Its default implementation uses the **get** method, discarding the other computed values, but more specific implementations can be much more efficient (see Section 9.4 on history managers). This method is useful to re-rank word lattices using the bunch mode and feature based LMs described below: LM states can be determined in a first stage and evaluated with the bunch mode afterwards to speed up the LM queries.

Another method related to **get** and similar to the is **basicArcs** which returns an iterator useful to traverse the set of outgoing arcs. These arcs are called basic to distinguish them from reference transitions used in RTNs and described in Section 9.1.3. This method is similar to the **arcs** method that can be found in other automaton based inter-

8 It is guaranteed to produce at most one result when the underlying **LMModel** is deterministic (indeed, some LM types are guaranteed to produce always one result). This special case could be taken into account in some cases.

9 The **Burden** value includes two different identifiers: a key and a word identifier. These values, together with the LM key and the score constitute the basic information to represent an ongoing decoding hypothesis. A consolidated hypothesis can be obtained afterwards from the previous hypothesis (referenced by the key stored in the Burden) and the values of the ongoing hypothesis. More details in Chapters 10 and 12.

10 There is not distinction between input and output labels since this interface is not designed for transducers. Regarding null transitions, although not directly exposed by the API, can also be partially tackled by means of non-determinism or by using a special word identifier to represent the empty symbol.

11 Our work is specially focused/biased towards dynamic decoding techniques.

faces. It is proposed for completeness since the previous `get` method is better suited for our decoders.[12]

As with previously described interfaces, there is a method to obtain the initial state of the model (**`getInitialKey`**) and another one (**`getFinalScore`**) to determine the probability of each LM state to be a final state.[13]

There exist two variants of **`getBestProb`** method:

- obtain the best score associated to the entire model. This value is pre-computed for automata representations (e.g. a count based n-gram converted into an automaton);
- retrieve the best score which can be achieved from a given **`Key`**.

Although this could be used to implement the LM look-ahead (LMLA) pruning technique, it would be quite inefficient.[14]

Pruning techniques exposed in the interface

As can be observed, the previous **`get`** and **`getFinalScore`** methods contain a threshold value as argument in order to apply a pruning during the generation of the transitions. Let us observe that if this pruning criterion were not available in the LM, the decoder would prune the resulting hypothesis in many cases. In this way, the decoder or, in general, the user of the LM, may compute the maximal LM score which would lead to a non pruned hypothesis by discounting from the decoder beam the partial score of the ongoing hypothesis before the LM contribution.

Despite the fact that this criterion could be easily removed (it can be applied afterwards in any case), it allows some improvements in many cases, specially when the set of destination states is quite high. For instance: a LM including edition operations may generate many destination states due to deletions.

Let us point out another important reason to prune LM destination states before retrieving them to the decoder: the representation of the destination LM key is a numeric identifier. This value can be just a reference to an automaton state statically stored in memory, but in other cases (NN LMs and, in general, when using feature-based LMs with a history manager described in Section 9.4) the resulting LM key is an index to a new LM context whose construction has a computational cost that can be skipped in this way.[15]

12 They are based on a lexicon estimator (see Chapter 10) to detect words from the observed signal (acoustic, optic,...) evidence in order to be checked against the LM afterwards. See the first paragraph of Section 6.10 for a brief review of other possibilities. Moreover, the usefulness of this method is quite limited when using smoothed LMs: the list of arcs may correspond to the entire vocabulary since the LM interface hide internal details such as modeling back-off n-grams by means of null transitions.

13 An additional method **`getZeroKey`** has been included for the special case of LMs (such as n-grams with back-off, described into more detail in Section 9.2) where there exists a lower order state not to be confused with the initial state. The method returns **`false`** when this state has not sense in the current LM. This method is used to ignore the context cues in LMs already trained with them. More details in the experimental part (the third part of this work).

14 We have not implemented for the moment features to support the more generic variants of LMLA since we have only tried the unigram smearing approach (see Section 10.6.7).

15 A previous version of the LM interface, which did not include the pruning criterion, returned "futures" to the destination key. A "future" is a concept widely used in

The use of a simple threshold value as a pruning criterion is perfectly suitable for a single query to a deterministic (or nearly deterministic) LM. However, there are other cases when a more sophisticate technique seems more appropriate. For instance: when the number of computed destination LM states can be quite high or when using the bunch mode (described below) to perform lots of LM queries simultaneously. In those cases, LM methods might receive a pruning object capable of more sophisticated technique (such as histogram pruning) as those described in Section 10.6.1, in order to apply pruning during the generation of the transitions. This extension has not been included for the moment, but it is obviously more general since it subsumes the single threshold criterion (which is a particular case). In any case, when the pruning technique is not required, it can be emulated by using as threshold a nearly zero[16] value or by using a dummy (which never prunes) pruning object.

Bunch mode

joint queries

A very useful feature to improve the performance of some LM representations and decoders proposed in this work is the capability of performing multiple LM queries. We have dubbed this extension "bunch mode" in analogy with a related neural network technique to evaluate the forward step of several inputs at the same time, taking profit of modern CPU capabilities exploited by linear algebra libraries.[17]

The use of NN LMs clearly illustrates the interest on grouping several LM queries associated to the same LM context, since they share the same forward evaluation. Although grouping several NN LM queries in this way to reduce the number of forward evaluations has been reported elsewhere (e.g. [Schwenk 2007; Sect. 2.1.3]), we are not aware of any LM interface description supporting this feature. An alternative to avoid bunch LM queries for NN LMs would consist in storing not only the computed values in a LM cache, for future uses, but also the scores of *all* possible words associated to the same LM context, which seems unfeasible.[18] The use of multiple LM queries can not only be used for grouping several queries associated to the same forward computation of the neural network but also to process several forwards at the same time, hence the name bunch mode.

Besides improving the evaluation of NN LMs, we will show in Section 9.3.1 that it can also speed-up the proposed count based n-gram representation.

The following additional methods of the **`LMInterface`** class are related with bunch LM queries:

Computer Science to represent a value to be computed. Although this concept is commonly employed to provide synchronization mechanisms in concurrent processes (See `http://en.wikipedia.org/wiki/Futures_and_promises`) it is used here to allow an "on demand computation" of the actual "Key" value. The use of the pruning criterion, combined with the bunch mode described below, make this feature unnecessary.

16 A log-probability C++ class is used in our implementations and a near zero value is used instead of a pure zero since working with logarithms.

17 This and other ideas are described into more detail in Section 9.5.

18 A more reasonable approach would consist in storing the last hidden layer, much tinier, and the softmax normalization constant, quite expensive to compute.

```
void clearQueries();
void insertQuery(Key key, WordType word, Burden burden,
                 Score threshold);
void insertQueries(Key key, int32_t id_key,
                   Vector<WordIdScoreTuple> words,
                   bool is_sorted=false);
const vector<KeyScoreBurdenTuple> &getQueries();
```

This API requires using several methods to perform the joint evaluation. This is one of the reasons why this class is detached from the **LMModel** representation. The usual procedure is as follows:

1. In a first step, **clearQueries** must be used before inserting multiple queries. It is used to indicate the **LMInterface** to clear the list of previous queries and to bring it the possibility of preparing the required data structures, which depends on the actual implementation;
2. one or several combinations of **insertQuery** and **insertQueries** methods can be used to introduce queries to be evaluated, they can be evaluated in this moment or delayed to the moment the results are requested, the latter method is useful when several queries share the same LM context but are applied to different words, its default implementation relies on the former method;
3. the **getQueries** method is used to retrieve the results. Depending on the implementation, it can perform the actual computation or just return the previously computed queries.

Let us observe that the result of possibly several LM queries are returned in the same vector. The reader may wonder how the results of each different query can be distinguished among them. Here is where the "burden" value enter the scene: this value can be considered as an identifier and, since it is copied from the origin LM query to their corresponding results, it suffices to use different values to be able to associate the results to their corresponding individual queries: If the user of the LM bunch mode wants to retrieve the answers associated to a given query, it suffices to select (from the overall resulting vector) those results which have the associated burden value. This might seem inefficient, but there is an explanation: this interface has been designed to simplify the decoders described in Chapter 12 which, in turn, use the ideas from lexicon decoders proposed in Chapter 10. In those decoders, the entire set of LM keys was stored in a contiguous vector using a "double ordering" so that LM keys of contiguous hypotheses were placed adjacent. In this way, the bunch mode returns a vector which can be used as is without resorting to the individual extraction of their constituents.

A reasonable procedure to implement bunch LM queries might depend on the cost of LM lookups which, in turn, depend on the LM type. For instance: automaton based representations are usually much faster to evaluate than NN LMs. In the first case, we can simply rely on the basic **get** method in order to implement **insertQuery** and **insertQueries** methods as mere wrappers which push back the re-

sults in an internal vector to be retrieved by **getQueries**. The case of more costly LMs, as is the case of NN LMs, could proceed as follows:

- when a query is inserted by means of **insertQuery**, it is stored in a data structure which allows us to group all queries related to the same LM context. It might be also desirable to obtain the set of words associated to each LM context and, in some cases, to sort them (which is the reason why the **insertQueries** method has the **is_sorted** flag parameter);
- the neural network computation is delayed until **getQueries** is called and the queries are grouped by LM contexts and are evaluated in groups of a configurable bunch size. All the queries of a given forward are computed at the same time and, in this way, no LM cache is required (although it could be used in any case).

From this last example we can observe that the role of the **clearQueries** method is not limited to clear the internal vector used to retrieve the results, but also to empty the data structure to group different queries by their LM context or key.

To summarize, the bunch mode has a default implementation based on the elementary **get** method, so it does not suppose an additional effort unless seeking extra efficiency. It has been co-designed with segment estimators and decoders in such a way that a sole **LMInterface** method call suffices to compute the entire set of LM look-ups for performing an entire Viterbi step in a one pass decoder. Hopefully, the availability of all these information, together with information about how to prune less promising results, would allow LM implementations to try to do "their best" to reduce the overall cost. We have briefly sketched how this can be used in a NN LM, more details and the additional improvements which can be achieved in automaton based representations are described below in this chapter.

Working with pre-determined vocabularies

We can also mention here another extension related to LM interfaces which, unfortunately, has not been implemented for the moment. We are quite confident on the novelty of this idea[19] which consists in precomputing a finite number of lexicon subsets which are organized in a lattice algebraic structure.[20] These lexicon subsets can be used to implement lexicon decoders which are more specific (and smaller) than the one covering the entire lexicon. The extension we seek should allow us to efficiently compute the minimal lexicon subset capable of covering all the words which can be used by the current set of hypothesis which, in turn, depends on their corresponding LM states. The proposed **LMInterface** method for this purpose could be as follows:

```
LexiconSubset minimalLexicon(Key key, Score threshold);
```

19 I have not been able to find it in other works and, although I think that the list of bibliographic references of this writing is "reasonably" large, this is negligible compared with the current number of papers. I have usually employed search engines to look for relevant keywords but this procedure does not constitute a guaranty. Moreover, this work becomes outdated really fast: *while it is being written*.

20 For instance, in a tree where each parent contains all the descendants and the root is the entire lexicon.

where **LexiconSubset** can be considered as an identifier to one of the lexicon subsets previously computed. The threshold parameter[21] can be used to refine the process by allowing the LM interface to discard those words which are only reachable from LM transitions that would be pruned anyway.

This method should be applied to each LM key appearing in an active hypothesis of the decoder at a given moment, which leads to a lot of results of type **LexiconSubset** (an integer identifier). In order to obtain the minimal lexicon subset which guarantees to produce at least all the words useful by all possible LM keys, it suffices to solve the lowest common ancestor (LCA) over the resulting lexicon subsets. This procedure is linear with the number of LM keys since the LCA of two subsets can be computed in constant time. A naive implementation of this method would either return the overall lexicon or to precompute a proper lexicon subset for each LM state in an automaton based representation, hence discarding the threshold value. We do not enter into more detail because, unfortunately, this feature is not useful when using smoothed n-grams (as is usually the case).[22] This feature is described elsewhere in this work, not only in Section 6.10.1 but also:[23]

- the word graph generation algorithm described in Chapter 11 might receive feedback information from the decoder which is using its output. This information may also include the most specific sub-lexicon required to pursue the process in order to allow the graph generator to use the most efficient lexicon decoder;
- the DAG decoder explained in Section **??**, used by the two stage decoder, may produce a feedback information (for the graph generator described before) including this information.

9.1.3 RTN interface

This section explains how the previous LM API can be extended to deal with RTNs. RTNs were introduced in Section 5.4.3 as a representation of a normal form for CFGs (a restricted case of the 2NF) rather than as described in their original presentation [Woods 1970] or in most other works.[24] Since the RTNs described in Chapter 5 have many resemblances with FSAs, it is not surprising to try to extend the former LM interface to deal with them. Moreover, we will propose an API suitable for a useful RTN extension which allows the use of different reference transitions starting at the same state, as described in Section 5.7.

21 A different threshold can be applied to each key by determining the proper threshold from the hypothesis score associated to the LM key.

22 For these models, the entire lexicon is almost always reachable from any LM state, leading always to the entire lexicon decoder unless taking into account the threshold parameter. Semantically restricted domain tasks might benefit from this technique.

23 As can be observed, this feature seems restricted to dynamic decoders. We believe that this is not a crucial drawback since other popular techniques, such as LMLA, suffer from this limitation as well (LMLA is not required in the static transducer composition approach because it is a particular case of weight pushing).

24 In those works, RTNs are a formalism related but different from CFGs and are usually composed by a set of disjoint finite state networks with rather complex procedural descriptions similar to subroutine calls. In many cases, the references to other models are placed on the nodes (Moore) rather than on the edges (Mealy).

basic vs reference transitions

This API should be appropriate for different types of decoding (e.g. based on the dynamic expansion, memoization and so on). These decoders usually require to detect when a given LM state has outgoing transitions labeled with references to other models rather than just being labeled with a word of the lexicon. They are called *reference* transitions as opposed to the former *basic* transitions.

Let us remark that reference transitions are processed differently from the basic ones: while dynamic decoders are usually guided by the evidence obtained from the observed signal (the set of words detected by a lexicon estimator), all the reference transitions outgoing a given state must be processed when this state is active. To this end, a method to iterate over the set of outgoing reference transitions is proposed:

```
ReferenceArcIterator referenceArcs(Key key);
```

The reference transitions reachable by these iterators contain:

- a unique reference arc identifier (a value of type **`ArcID`**). Observe that by "unique" we are meaning that each reference transition has a different identifier even if they use the same underlying model. It can be useful both for decoders using the dynamic expansion and the memoization approaches;
- a first transition probability;
- the state the arc is referencing.

Note that, since there can be several destination states associated to this reference transition, only the arc identifier and the initial state are required at this point:

- the arc identifier can be used to construct a memoized spaghetti stack to associate unique identifiers to every arbitrary sequence of arc expansions (see Chapter12);
- the initial state and the position in the input DAG can be used to identify a parsing context where all arc identifiers (together with their corresponding contexts) are stored. They are retrieved later when the associated model reaches a final state.

In any case, the active hypothesis of the decoder contains, at least:

- the current **`Key`**;
- the associated context identifier (a pointer to an element of the spaghetti stack in the dynamic expansion version, an index locating the pair (initial state, initial input vertex) in the tabular version).

Other fields of active hypothesis are described into more detail elsewhere (not only but, specially, in Chapter 12). What is important, in this moment, is to observe that we can obtain the set of active ArcIDs associated to the context identifier.

Since the RTN extension we are implementing allows submodels with an entry point and possibly several exit points, the method to determine if a given state is final has the following profile:

```
ArcDestIterator isFinal(Key key, ArcID arc);
```

This method detects if the **key** argument is a final state and returns a value allowing us to iterate over the list of destination states associated to the arc given by the second argument. Besides the destination state, the transition probability can also be obtained from the **ArcDestIterator**. Note that a first transition probability was already obtained by means of **referenceArcs**. This first probability was common to all destination states and was computed as the maximal transition probability of all exit points associated to the arc, which is a rudimentary way of applying weight pushing.

We can also observe that the use of such an iterator allows a single reference transition to have several destination states associated to different exit points. A notable example of this feature is the possibility of using a tree lexicon model with as many exist points as lexicon words. Observe also that the probabilities are associated to the reference transition rather than to the underlying model, which increases the possibilities of sharing or reusing the models.

9.1.4 Heterogeneous lattice LM interface

This interface is the counterpart of the "Heterogeneous lattice LM" described in Section 6.11. Departing from the idea that LM histories can be organized in the form of a lattice structure [Dupont and Rosenfeld 1997; Bilmes and Kirchhoff 2003], we showed how this idea can be extended to organize LM history managers, external context managers, feature extractors (e.g. topic estimation) and even LM combination operators to construct "feature based LMs" (feature based n-grams being a notable case) as a LM node on the top of a DAG and even to manipulate this structure (e.g. to infer the structure from the set of required features, simplify them, etc.).

manipulating internal components

Since these are preliminary ideas, we will only sketch the basic ideas to point out their importance. The former interfaces are valid and we only would need to include new APIs for other parts of the design and to structure them in a suitable way in terms of inheritance. We have not found similar words in the literature although some lines in this direction can be found. For instance, the class hierarchy of Moses framework [Koehn *et al.* 2007] makes the LM base class as a subclass of "stateful feature function", which is a general class for those feature extraction functions that are able to take a state to obtain a new one by means of a transition.

9.2 COUNT BASED n-GRAM IMPLEMENTATIONS

Fast LM look-ups and a reduced spatial footprint are the main desirable features of LM representations. Accuracy also enters into this equation, in some cases, when quantization methods are used to reduce the size of LM scores. As count based n-grams are the most popular and widespread type of LM, it is not surprising to find a lot of representations for this type of LM. Note that the most suitable representations for LM estimation is not necessarily the most appropriate

for decoding.[25] This section is focused on representations for decoding purposes. These representations can be classified by different points of view, but we will distinguish between:

- explicitly representing n-grams; and
- converting them into weighted automata.

The first one seems more suitable for n-gram LM interfaces. The second one appears more general and is the one we have chosen for our proposed representation.

orthogonal features

Many representation issues are common to both representation approaches, as is the case of: memory mapping, the use of a cache for accessing the last queries, or the quantization of scores to reduce memory usage, to mention a few. Another interesting orthogonal issue to take into account is whether the representations are thread safe or not.[26] The use of cache poses synchronization problems and some systems use a separate cache for every thread. Memory mapped LMs can even be shared by several processors on the same machine in some cases.

Explicitly representing n-grams

implicit dense representations

There are straightforward representations specially tailored for lower order n-grams: unigrams can be represented by means of a simple vector indexed by word identifiers. Bigrams, for small vocabulary tasks, can also be represented by means of matrices. For higher order n-grams (and even for bigrams), such a dense or fully connected representation is highly inefficient. Although n-gram sequences are not stored in those representations (only their values), the set of n-grams observed in the training data is just a tiny fraction of possible word combinations, making necessary the use of sparse representations.

`arpa` *format*

Most n-gram models are based on back-off for smoothing. This is taken into account in most representations which also store lower order n-grams together with back-off weights. Indeed, the standard *(de facto)* file format representation for count based n-grams is the "`arpa`" file format produced, at least, by SRILM [Stolcke 2002], IRSTLM [Federico *et al.* 2008] and openFST[27] toolkits. In this model, for an n-th order n-gram, the set of unigrams, ..., $n-1$-grams are stored together with their smoothed probabilities and back-off weights. Higher order n-grams are stored with probabilities but not with back-off weights.

trie is the standard choice

The data structure of choice for many implementations is the trie, also known as prefix tree. Trie nodes are usually labeled with word identifiers and the trie depth is the order of the n-gram. Among the LM toolkits which use tries in some of their representations we can mention CMU SLM [Clarkson and Rosenfeld 1997], SRILM, IRSTLM, MIT LM [Hsu and Glass 2008], BerkeleyLM [Pauls and Klein 2011] or kenLM [Heafield 2011].

25 Usually, the memory footprint for estimation is higher than for decoding. In order to construct a count based n-gram model, it is required to count n-gram occurrences in a training corpus to estimate the smoothing parameters [Chen and Goodman 1998].

26 Related with that, remember that we have chosen to separate the **`LMModel`** and the **`LMInterface`** classes in order to easy the implementation of thread safe LMs.

27 `http://openfst.cs.nyu.edu/twiki/bin/view/GRM/NGramPrint`

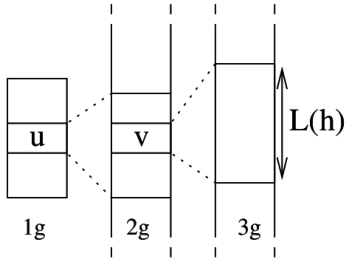


Figure 113: Figure taken from [Massonie *et al.* 2005; Fig. 2] illustrating a count based n-gram represented as a tree of word-lists up to 3-grams. $L(h)$ is the list of possible words and scores for LM history (u, v).

Trie based representations may differ in the order n-grams are stored[28] and in how the trie is represented in memory. Relating this last issue, there are several possibilities:

- an array of entries sorted by the vocabulary identifier, as depicted in Figure 113. Note that a sole array is required for each n-gram order. Words can be searched in the sorted array by means of binary search or also by interpolation search [Perl *et al.* 1978]. The data accompanying each word identifier is the first index[29] of the next order, excepting the last table [Massonie *et al.* 2005]. Many LM libraries support this representation either as default (IRSTLM, MIT LM) or as a *sorted/compact variant* (SRILM, BerkeleyLM);
- a hash table within each trie node (e.g. SRILM or BerkeleyLM);
- a global hash table for all trie nodes, or at least for all n-grams of the same order, as is the case of KenLM [Heafield 2011].

Representations based on hashing usually employ open-addressing schemes. Some implementations rely on perfect hashing[30] [Cardenal-Lopez *et al.* 2002; Zhang and Zhao 2002; Li and Zhao 2007; Huijbregts 2008]. For instance, in [Huijbregts 2008] a perfect hash table is used for each n-gram order. Perfect hash tables require to know the set of keys in advance to obtain the hash function. Keys have to be stored in the table because queries to non-existing keys are indexed to random positions. Another use of perfect hashing for LM representation is proposed in [Daciuk and van Noord 2004] where perfect hash automata are used to assign a unique identifier to each word sequence.

perfect hashing

Some representations, as is the case of arrays of sorted entries or the use of hashing methods based on perfect hashing, do not allow efficient updating involving insertion or deletion of trie nodes. A mutable implementation is mandatory when managing LM histories for feature based n-grams described in Section 9.4.

immutable representations

An inconvenient of trie representations is that a look-up is required for each visited node, which increases with n-gram order, not to mention backing-off. This can be avoided by using the automata representation of back-off smoothed n-grams described below. Note also that some LM implementations manage an internal context making it possible to easily access a context from a simple identifier, which is an

28 Forward or backward: in backward representations the last word of an n-gram are looked up first. The advantage of backward order is that, when a transition is not found, we have obtained the longest stored suffix for free.

29 The use of indexes has many advantages over using pointers.

30 http://en.wikipedia.org/wiki/Perfect_hashing

alternative (although more cumbersome, from our point of view) to the automata representation. In this case, the trie representation may include additional pointers to easily access the suffix of each trie node. Some representations store the entire n-gram (avoiding tries) in sorted arrays of in hash tables to perform a sole look-up per n-gram.[31]

Since some LMs are really huge, the memory usage is of primary importance. It is possible to prune and reduce the LM itself [Stolcke 2000a]. Some representations are focused on efficient spatial representation even if it is at the expense of sacrificing performance or accuracy. Relating accuracy, the size of nodes is often reduced by using quantization on scores. [Watanabe *et al.* 2009] proposes a non-loss method which combines the trie representation with variable length coding and block-wise compression. Other toolkits simply assign less bits to the mantissa or even remove the sign bit.[32] Others store all scores in a separate vector removing duplicates and reducing the set of values by clustering. Memory is reduced because indexes occupy less bits than scores themselves.

randomized LMs

There are also lossy compression representations based on randomized probabilistic data structures such as Bloom filters[33] in such a way that n-gram keys are not explicitly stored at the risk of false positives [Talbot and Osborne 2007; Talbot and Brants 2008].

A different way to tackle the problem of huge LMs, which is compatible with previous approaches, is the use of secondary memory and the possibility of distributing the LM among several computers. The use of secondary memory in the form of memory mapping is discussed below. In order to alleviate the higher cost of some storage efficient representations, a cache can be used for last used LM histories.

Representing n-grams by means of WFSA

from `arpa` to WFSA

Storing a n-gram model with back-off smoothing[34] by means of deterministic WFSA is a bad idea because its size grows exponentially with the n-gram order. This problem is not found when using null transitions to emulate back-off transitions. The basic approach to construct such an automaton from the set of n-grams of a n-th order n-gram LM (e.g. as stored in the `arpa` file format) is as follows:

- construct an automaton state associated to each n-gram of an order lower than n. For instance, for a trigram LM, unigram "a" is associated an state, also bigram "ab", but not trigram "bca";
- create a final state;
- the initial state is the (unigram) <BOS> context cue;
- each n-gram with an order lower than n is also used to construct a transition from the state associated to its prefix to its own state. For instance, unigram "a" would lead to a transition from the zero-gram to the state "a", another transition from "a" to "ab" would be created from bigram "ab", etc.

31 Back-off would require additional look-ups.

32 For log-scaled probabilities associated to transitions, but not for back-off scores which can be positive in some cases.

33 `http://en.wikipedia.org/wiki/Bloom_filter`

34 Linear interpolation can be emulated in an n-gram model based on back-off.

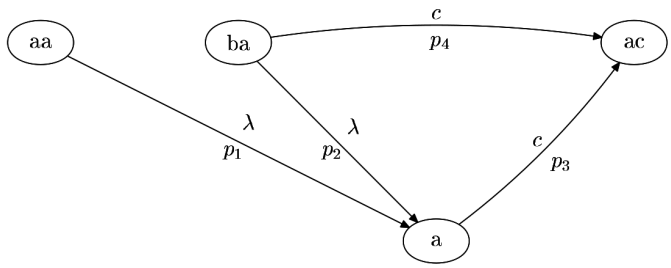


Figure 114: Figure taken from [Llorens Piñana 2000; Fig 4.3] showing why the compact automata representation of a back-off smoothed n-gram is not equivalent "in the Viterbi sense" to the expanded representation if we treat failure transitions as standard null transitions (here depicted with the λ symbol). If p_{aa}, p_{ba} and p_{ac} are the respective probabilities of states "aa", "ba" and "ac", a problem with null transition p_2 occurs when $p_2 p_3 > p_4$.

- higher order n-grams are only used to construct transitions. For instance: "abc" would lead to a transition from "ab" to "bc";
- create a null[35] transition from each state to its prefix (initial, final and zero-gram states are excluded);
- create a transition to the final state from each n-gram whose final symbol is the <EOS> context cue.

Indeed, as pointed out in [Llorens Piñana 2000; Section 4.4.2], this model is not equivalent to the actual (deterministic) back-off smoothed n-gram, as illustrated in Figure 114. Quoting[36] this work:

> *However, this automaton definition extension cannot be applied to speech recognition problems where we do not know which word starts in a given state until we have evaluated them (the same happens with back-off).*

it would seem that only an approximation of the true backed-off models can be used. The author also cite the most common approaches to tackle this problem, namely:

- using the non deterministic automaton as an approximation of the correct model [Riccardi *et al.* 1995];
- dynamically expand the deterministic automaton when using back-off [Bordel *et al.* 1997; Torres and Varona 2001].

obvious solution

But this problem has a much more obvious solution: Do not interpret those back-off transitions as null transitions in the way they are defined in formal language theory, but use them only when the transition labeled with the word is not found.

failure transitions

This trick, which is probably discovered and used in practice in many systems (as is our case), has been rediscovered[37] in [Allauzen

35 Or, rather, a *failure* transition, as described below. We can also remark that our arpa to automaton conversion removes useless states (non-final states with fan-out=0) and the transitions incoming to them are redirected to their proper back-off states.

36 Spanish original: *"Sin embargo, esta extensión de la definición de autómata no puede aplicarse en problemas como reconocimiento del habla donde no sabemos qué palabra parte de un estado hasta que las hemos evaluado todas (lo mismo sucede con el análisis sintáctico por back-off)"*.

37 It is sad and unfortunate to discover that this work does not cite any prior art relating the use of null transitions to represent those models as is the case of k-testable languages or the works of [Riccardi *et al.* 1995; Bordel *et al.* 1997; Llorens Piñana 2000].

et al. 2003] where authors named those back-off transitions *failure transitions* to distinguish them from null transitions. More recently, [Roark *et al.* 2011] explained how those back-off transitions can be specified by means of null transitions using a lexicographic semiring, hence removing the requirements of the ad hoc "otherwise" semantics.

It is possible to provide a count based back-off smoothed n-gram implementation for the automata based interface either by using the "otherwise" semantics or with the use of the lexicographic semiring. The difference is hidden from the outside, but the last approach allows reusing a general toolkit for finite state automata/transducers able to deal with different semiring types (e.g. [Allauzen *et al.* 2007]). The use of meta-programming facilities, such as C++ templates, allows the generality of the semiring approach without sacrificing performance.

From the decoding point of view, the automata representations of a n-gram LM can be managed more efficiently than the explicit n-gram representation counterpart. LM look-ups are faster since, excepting back-off transitions, only one search per LM look-up is required. Using trie based n-gram representations with the n-gram interface, as searches as the trie depth are required. Note that the automaton can also be used with the n-gram based interface. It suffices then to perform $n - 1$ queries, departing from the state associated to the zero-gram, to retrieve the state corresponding to the context and another query to retrieve the LM transition score. General issues relating the representation and storage of automata, not only for n-gram equivalents but for the general case, are delayed to next section.

9.3 AUTOMATA REPRESENTATION

keep it simple

There exists different alternative ways of statically representing automata but, in essence, they are a particular case of graph representations [Cormen *et al.* 2009]. The automaton representations used in most cases in this work are mainly used for Viterbi and Forward algorithms. We only need to obtain the set of transitions outgoing from a given state, but not the incoming transitions. In this case, there are two main alternatives, namely: adjacency matrices[38] and adjacency lists.[39] Since automata are usually sparse graphs, the adjacency lists seems a natural choice. Indeed, given that automata are not mutable, it is possible to compact the set of arcs in a vector so that each automaton is represented by means of two vectors:

- one containing all the transitions sorted by origin state (transitions outgoing from the same origin are stored together);
- another of size the number of states.[40] Every component contains an index of the first transition in the former vector.[41]

A representation more versatile and suitable for figuring out incoming transitions is the vector of transitions [Allauzen *et al.* 2007; Section 3].

38 `http://en.wikipedia.org/wiki/Adjacency_matrix`
39 `http://en.wikipedia.org/wiki/Adjacency_list`
40 Plus one, since a sentinel value is required.
41 The number of transitions is deduced from the first transition of the next state.

9.3.1 Proposed "fan-out based" representation

Our representation is based on the automaton representation which used two vectors (one for states and other for transitions) described before. This representation is used to store WFSA constructed from `arpa` LMs as described in Section 9.2. Let us remark that this is not a general automaton representation since, although each state has an optional failure transition, general null transitions are not considered.

Since different automata representation alternatives may be beneficial in some particular cases, we believe that it is convenient to measure certain properties on the LMs used in this work. The following conclusions are, hence, only applicable to the extent that other LMs also display these properties. The key property that has allowed us to improve this straightforward representation is the following fact we have empirically observed (detailed later in Section 13.1):

> *The fan-out of states (number of outgoing transitions) in this type of automata is distributed in a very asymmetrical way: most states have a very low (even zero) fan-out and only a few ones have a really huge fan-out.*

our fan-out based proposal

The proposal consists in associating state identifiers to states according to their fan-out. In this way, states with the same fan-out are stored contiguously. It is possible to compute a state identifier threshold to determine whether the fan-out of a given state if below another threshold or not. The main reason for this procedure is to speed up the search. Thresholds applied on state identifiers are used to select the best way to search a given transition:

- for fan-outs below a given threshold, a linear search produces less overhead than a binary search. Empirical measurements show that, for values below 20, linear search is faster. States with 20 or less outgoing transitions comprise (for the previous example) nearly 97 percent of states;
- in other case, a binary search is performed, although "interpolation search" [Perl *et al.* 1978] is a better choice for even larger values. This type of search has been recently proposed in [Heafield 2011], where it is shown that it is useful for vector sizes larger than 4096, although it its better to switch to binary search when the size decreases during the search;
- fan-outs larger than a given threshold could be represented by means of dense vectors.[42] This is not only faster, but also allows removing the word identifiers from transitions and avoid back-off transitions. Transition occupies 64 bits and can be reduced to 32 bits for those special cases, it is easy to determine which fan-out threshold can be used to choose this search technique as a function of vocabulary size.

implicit dense representations

Sorting the states by their fan-outs has another advantage: it allows us to compute analytically the index of the (consecutive) set of transitions associated to a given state. To this end, we use an auxiliary table of size the number of different fan-outs appearing in the automata:

42 Word labels without transitions could be marked with a special score, but it is obviously much better to pre-compute the value that a back-off transition would lead.

```
struct LinearSearchInfo {
  unsigned int first_state;
  unsigned int fan_out;
  unsigned int first_index;
};
```

The size of this table is usually very small. For instance, the model from the previous example has 3 230 247 states but they are grouped in only 1885 different "types of fan-out". We only store the $N = 20$ first classes. If we had used a "first transition" field for every state, we would have needed around 12 Mbytes of memory[43] whereas the proposed table has a size of 240 bytes. A simple integer comparison determines whether the array of first transitions has to be used or not. Other particular features of our LM representation are:

- back-off failure transitions are stored in states, not in the vector of transitions, because they are not used as regular transitions, these transitions are not exposed to the LM interface API and are used transparently by the **`get`** method;
- the best outgoing transition is stored for each state, which is used for one of the variants of **`getBestProb`** method;
- it is possible to store transition scores as a ratio of the best outgoing score. In this way, the number of bits required for each transition can be reduced. We have not reduced the number of bits because each transition occupies 64 bits. This simplifies the access to the vector and the size of our models is not so huge.[44]

LM-queries polices

The speed of a LM representation can be measured by the average number of LM look-ups per second. This may vary a lot depending on the use of LMs, notably on the policy of queries. Most decoding algorithms usually perform consecutive search transitions for a given LM state, but sometimes they are independent queries. When decoders are able to pack the entire set of LM queries from a given LM state (bunch mode) it is possible to take profit of the contiguous storage of arcs sorted by word identifier. Assuming that queries are sorted:

- the linear search applied for states with a low fan-out can be replaced by a version of the merge algorithm (the auxiliary algorithm used in mergesort) which computes the intersection of two ordered sets;
- the binary search technique can also take profit of a series of ascending values since the value found in one search can be used as lower position of next one. Depending on the number of queries, it can be convenient to change to the previous linear search.

In general, several searches on the same memory region increases locality and may reduce the number of cache misses. This cache of the

43 Using 32 bit integers.

44 LM size of models recognition tasks is negligible compared to those of SMT tasks.

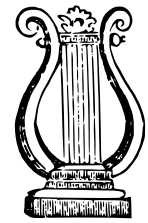
hardware memory hierarchy is not to be confused with an explicit LM cache. Relating the LM cache, the rationale for using it is quite old [Ravishankar 1996], as explained in [Daines 2011; Section 3.4.1]:

> *Since a given hypothesis word nearly always produces multiple word exits until its final phone is pruned away by the beam, it is possible to cache the LM and path score for that word.*

We have not implemented a LM cache, preferring to simplify the automaton representation to improve the *amortized* costs of several related queries and to adapt decoders to use a LM queries policy taking the best profit of it. Fortunately, NN LM implementations also benefit from this LM query policy.

LM and decoder codesign

9.3.2 Memory mapping



In order to reduce the start-up time of our recognizers, we have stored our LMs in a format which basically mimics the internal representation described before. This representation, not designed to be shared with other toolkits, is usually generated from a LM in `arpa` format and has been named `lira`.[45]

It is possible to further reduce not only the start-up time but also the memory requirements by means of memory mapping. Read only data structures not only makes easier to get the thread safe property but also allows the LM representation to be saved in disk in a binary form which can be used as ordinary memory thanks to the **`mmap`** system call that can be found in most operating systems. The use of memory mapped LMs is widely reported in the language modeling literature [Heafield 2011] and has several advantages:

- memory mapping dramatically reduces the load time because parsing is skipped. Time is particularly reduced in the "lazy version" where memory pages are loaded on demand. There is a drawback, though, since this may produce a higher latency. In order to avoid page misses during decoding, we can force the loading of pages by reading the entire model before decoding;[46]
- it may also reduce the memory requirements, since it is possible to load LMs larger than the system memory (provided this size is addressable from the virtual memory).

and, in particular, when this mapping is done with **`mmap`**:

- the memory usage is managed by the operating system page-cache mechanism. It usually profits the physical memory not currently used by any process to store these pages. If the computer load augments, this part of the memory is released transparently and in a different way than the rest of the process (subject to memory swap), producing a degradation in performance;

45 It is not an acronym. A lira is a musical instrument like a little arpa. We wanted a more compact representation than `arpa`.

46 For instance, using the **`MAP_POPULATE`** flag.

- when several decoder processes use the same LM, some operating systems may reuse the mapped memory,[47] which would be shared among them.

The inclusion of this technique in the proposed LM representation has been straightforward because the representation is based on vectors without memory references. Since the `lira` format header contains all the information required to construct the set of vectors, it suffices to allocate them on a memory mapped memory region which is stored in a file together with the offsets required to retrieve each vector afterwards. We have just implemented a method to save the model in binary and another one for loading this binary using **`mmap`**.

9.3.3 Proposed RTN representation

The proposed RTN internal representation is intended to be used with the API extension described in Section 9.1.3. This interface allowed a novel and uncommon feature consisting in reference transitions with several exit points (arcs would hence resemble forks). The two basic methods offered by this interface are:

- **`referenceArcs`** allows us to traverse the set of reference arcs outgoing the given key;
- **`isFinal`** is used to locate the destination states when using a reference arc and an exit point (or final state).

As with the previously representation, we have to take into account the expected properties of the models that will probably be used. In this case, we expect as the common scenario that most of RTN states will have zero outgoing reference transitions. This means that using a dedicated, number of states sized, vector to store the first reference transition of each state is a waste of space.

We have also decided to implement the RTN representation as an extension of the `lira` model described in previous section. This means managing less representation types, which help to reduce the overall complexity and simplifies maintenance. Although the main goal is to include the new features in the most efficient way, we have also a hard restriction: to involve no extra cost when using standard back-off smoothed count based n-grams.

In order to fulfill this last requirement, all the information related to reference transitions will be preferably stored in different data structures. Let us remember that the `lira` representation assumed that states were sorted by their fan-out. This fan-out will be limited to basic transitions. This makes it difficult to use the state identifier to code the presence of RTN features. However, since we are usually using 32 bit integers to represent states and 30 bits suffice, we can reserve two bits of the state identifier to: 1) determine whether the state is an exit point or not, and 2) to detect when a given state has outgoing reference transitions.[48]

47 Mapped memory can be accessed by child processes with **`MAP_ANONYMOUS|MAP_SHARED`** flags. **`shm_open`** with **`mmap`** would allow sharing data between unrelated processes.

48 Note that the use of these bits is not mandatory. Although the API could be implemented without them, they can improve the RTN methods performance. However, this

Each reference transition is assigned a unique numeric identifier, but we are free to design this arrangement. The following one is suitable to efficiently implement the desired methods:

- reference arc identifiers can be used to index a vector (contiguous values starting at zero);
- the set of reference arcs outgoing the same origin state are assigned consecutive identifiers;
- arcs are assigned following the numerical state/key identifiers.

This numbering scheme, illustrated in the example of Figure 115, allows us to easily locate the set of reference transitions outgoing a given state. They are required by the iterator returned by:

```
ReferenceArcIterator referenceArcs(Key key);
```

A vector, called **`iterArcVector`**, with the following fields:

- the state identifier;
- the identifier of the first reference arc outgoing this state.

is used to this end. The size of this vector is the number of "states with some outgoing reference arc", which is usually much lower than the number of states (otherwise, a "first reference transition" vector is better). This vector is sorted by state identifier, which allows us to efficiently locate the first arc associated to a given state.[49] The identifier of the last arc is implicitly given by the first arc identifier at the following vector position. With them, it is trivial to implement the iterator returned by **`referenceArcs`**. It suffices to use the reference arc identifiers in the corresponding range to access a vector (named **`referenceArcVector`**) with the following fields:

- initial state of the referenced model (not to confuse with the origin state of the reference arc);
- the transition probability applied when entering into the model;
- the first index to a vector (called **`exitPointsVector`**) of the exit points associated to the reference arc. This last field is not required by **`referenceArcs`**, but is convenient for **`isFinal`**.

Relating the other method of the RTN interface extension, it is convenient to remember that the RTN version we are implementing assumes that any sub-model that can be referenced has just an entry point. However, there can be several exit points. Moreover, each different reference arc using the same underlying sub-model (i.e. leading to the same initial state of the referenced model) is free to use any subset of these exit points (as illustrated in Figure 115).

This is the reason why the method returns an iterator leading to zero, one or possibly several destination states, each one with an associated transition probability:

means that a bit mask or shift have to be ordinarily applied to destination states. An alternative consists in locating these bits in the vector used to store the back-off information, in a dedicated bit-vector or in bloom filters (false positives are not a problem).

49 Let us remark that the use of the bit to detect states with some outgoing reference transition can be used to avoid the search in most cases. Observe that the vector is also sorted by reference arc identifiers as well, which allows us to efficiently locate the origin state of a given reference transition, if needed.

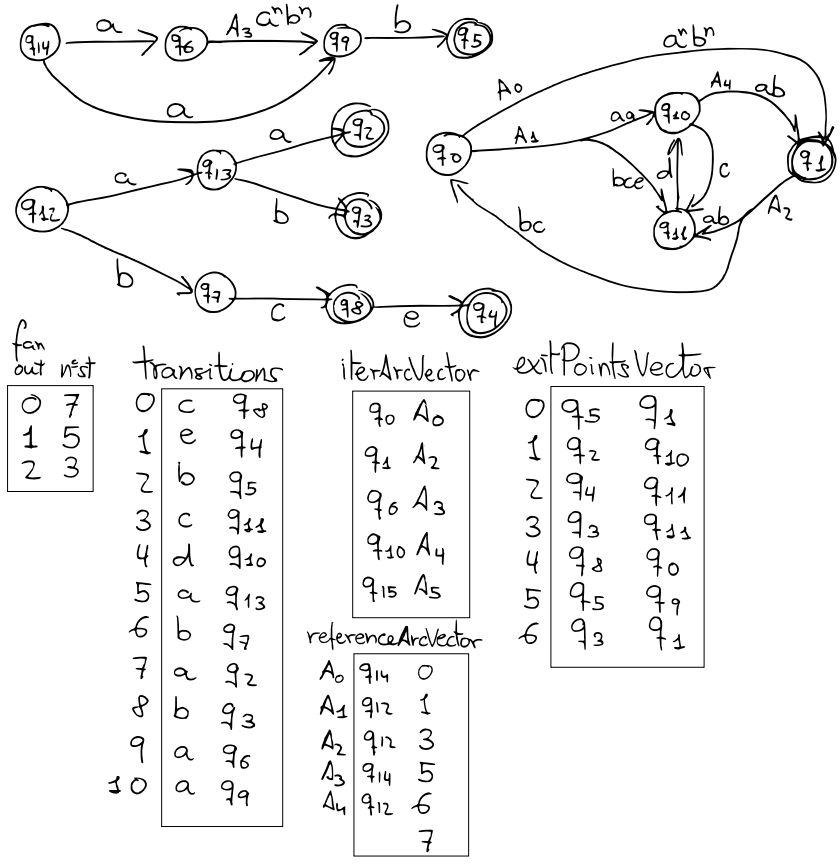


Figure 115: An example RTN representation. On top we can see a RTN with 15 states (although three sub-models can be appreciated, this distinction is not required). As we can observe, states are numbered by their fan-out (taking only basic transitions into account). The initial state of the overall model is q_0 which only has to reference transitions (A_0 and A_1). We can observe the presence of transitions with several exit points (e.g. A_0 can lead to q_{10} and to q_{11}). The sub-model starting at q_{14} is a simple example generating the language $\{a^n b^b | n \geqslant 1\}$, and the sub-model starting at q_{12} is a typical lexicon-tree. Below the graphical RTN representation we can observe the different tables described in Section 9.3.3. A table called **`firstTransition`** is not depicted since the maximal fan-out appearing in this example is 2 and this table is only used for states whose fan-out is bigger than a threshold (typically 10). Semiring weights are not shown to simplify the scheme.

```
ArcDestIterator isFinal(Key key, ArcID arc);
```

The **key** is a possible final state of the model referenced by the **arc**. This method is used just for a given arc when using the dynamic expansion approach[50] and may be several ones in the tabular decoding approach. In any case, the implementation of **isFinal** is as follows:

1. use the **arc** argument to locate its range of exit points. This range is obtained from the last field of the **referenceArcVector**; and
2. search the **key** argument value in the **exitPointsVector** in the range obtained in the previous step. Note that this value may appear zero, one, or several times.

The vector **exitPointsVector** has the following fields:

- the final state in the referenced sub-model;
- the destination state of the arc identifier;
- the corresponding transition probability.

and the range associated to each reference arc is sorted by the final state field. Relating the search in this vector, the same trick used to locate basic transitions, described in the previous section, can also be applied here: we can choose using linear search, binary search or interpolation search depending on the size of the sub-vector.

The example of Figure 115 will give a more precise idea of how the different vectors described so far can be used to represent RTNs in a form suitable for using the proposed API and leading to the previous `lira` model when no reference arcs are used.[51]

9.4 n-GRAM HISTORY MANAGER FOR FEATURE BASED n-GRAMS

This section is focused in "feature based n-gram models". Contrarily to count based n-grams, they estimate the probability of the following lexical item given a set of features extracted from the past history. NN LMs, their most prominent example, will be described in Section 9.5.

In order to use feature based models with the automaton LM interface, a data structure that we have called "history manager" is required: its main goal is to manage the set of n-gram histories (the last $n-1$ lexical items) associating a LM key or state to them and, reciprocally, to retrieve this past history from the key value. It is easy to extract the desired features from this set of items afterwards. Indeed, history managers could also be presented to the reader, in more general terms, as a wrapper for using the automaton interface with models designed for the n-gram interface.

interface wrapper

50 The arc is the last expansion, can be accessed in constant time from the decoding context of the active state. The context is a pointer to an spaghetti stack of arc identifiers.

51 Note that this representation, as with the basic `lira` model, is not suitable for directly specifying LMs but rather to be compiled from another format.

There exists many different data structure alternatives to associate a unique identifier to each possible vector representing a LM history, to check whether a particular LM history has already been appeared and, when necessary, to add it to the collection. The most obvious choice is the trie for the same reasons it is used to represent n-gram LMs (as described in Section 9.2). But not all n-gram trie representations supported insertion and removal of nodes. That is why we have decided to implement the history manager with a trie based on a global hash table to store its nodes: the hash table receives a tuple composed by a parent node and a label arc and returns the child node. In order to use this representation more efficiently with the automaton LM interface, an additional reference is stored, in each node, to directly access the node corresponding to its longest proper suffix.

garbage collection

It is convenient to include a new method, in the LM automaton interface, to specify which LM identifiers are not longer referenced by the decoder. This would allow releasing resources more easily. Another alternative is to populate the hash table and to release it after the entire input has been processed. We believe that it is better, for our purposes, to include a method on the LM interface where the entire set of keys used by a decoder is provided to the LM history manager so that it can remove unused nodes by means of *mark and sweep*.[52] This procedure can be applied when required, usually when the input data is too large. A simpler approach consist in just clearing the entire trie after each input sequence has been processed. Our current implementation uses a hash table with open addressing collision resolution where each node has a timestamp to easily detect which nodes are free. This timestamp can be as small as 1 byte, so that the entire trie vector has to be traversed for cleaning only one out of 256 times.

precomputed static trie

Note that it is also possible to mix a static and a dynamic trie in such a way that more probably used nodes are permanently stored (a kind of "ROM memory") and only new nodes are dynamically created and removed. This feature can be easily included by reserving a special timestamp value of zero to mark persistent nodes which allows us to have a predefined trie part with the most common keys.[53]

resizing

Unless we can bound the size of the trie, the hash table has to be resized. This constraints the choice of the hashing approach since index position cannot be used as LM keys when using open-addressing collision resolution unless using incremental resizing,[54] Another alternative would be to add a new field for the key value (instead of using the index vector) or change to separate-chaining.

Finally, note that a LM history manager can also be used not only to obtain the complete n-gram representation of a given LM identifier but also to store LM scores. This would make them to act as a LM cache not only for LM scores but also, for instance, for storing softmax normalization constants in the case of NN LMs as described in Section 9.5.

52 It is a very basic garbage collection technique divided into two stages: in a first step, the useful nodes are marked and, in a second step, the entire set is traversed removing the elements that were not marked in first place.

53 In this way, the vector has to be explicitly traversed for cleaning one out of 255 times.

54 A new empty table is used and the old one preserves its values and is not removed.

9.5 NEURAL NETWORK LM

This section addresses NN LMs. NN LMs were described in Section 6.6.1 and now more implementation details are addressed. This type of LM is much more recent than count based n-grams. Nevertheless, many interesting representation improvements have appeared (and are constantly appearing) in the literature. The basic architecture of such a model comprises:

- a projection layer [Bengio *et al.* 2003], with shared weights;
- hidden layers; and
- an output layer usually based on the softmax activation function (Equation 9.8).

The main approaches to speed-up the evaluation of these LMs comes from different fronts. There exists general techniques to speed up MLPs as is the use of bunch mode, linear algebra libraries to efficiently use hardware features, the use of GPUs, etc.

Relating techniques more specific for NN LMs, it is possible to reduce the cost of the input and the cost of the output layers. To reduce the cost of evaluation at the input layer, we can:

- replace the projection layer by table look-up on trained models;
- use the shortlist approach at the input layer. The use of the shortlist approach at the output layer was described in Section 6.3.3 and is widely reported in the literature, but it can also be used at the input layer as described in [Zamora-Martínez 2012];
- a novel idea based on the memorization of the partial sums of the first hidden layer, described below in Section 9.5.3.

But the most critical part is, by far, the output layer because it is much larger and because of the use of the softmax activation function. This function requires to compute the entire set of outputs no matter which ones are actually required (unless using unnormalized outputs, also discussed below). Relating the ways to speed up this part that we can found in the literature we can mention:

- the shortlist approach at the output, as described in Section 6.6.1, can be used to reduce the output layer to a fixed vocabulary size usually much lower than the overall vocabulary;
- structured output [Le *et al.* 2011] which basically consists of using a softmax layer to determine a word class and other softmax layers to determine the probability of a word given their class, we propose a novel extension of this idea in Section 9.5.4;
- memorization of softmax normalization constants to avoid the computation of the entire output layer on trained models [Zamora-Martínez *et al.* 2009a; Zamora-Martínez 2012], an improvement over this basic idea, based on training skipping NN LMs, is described in this section;
- it is also possible to convert the NNLM into a statistical back-off LM [Arisoy *et al.* 2013; Wang *et al.* 2013];

- the former approaches can be used to speed up the NN LMs during testing, but not during training. The use of the Noise-Contrastive Estimation technique introduced by [Gutmann and Hyvärinen 2010] (and first used for NNLMs by [Mnih and Teh 2012]) can be used to speed up the training phase by avoiding the computation of all the neurons of the output layer;
- some recent and promising works point to the possibility of completely avoiding softmax constants by means of training procedures aimed at reducing the variance [Shi *et al.* 2014];
- two different novel approaches are proposed in this work: 1) to estimate the softmax normalization constant by means of an auxiliary LM with, possibly, a much restricted vocabulary (this LM can easily be obtained by including a secondary softmax output of a much reduced size), and 2) adapting the distilling technique of MLPs [Hinton *et al.* 2014] to improve the training of NN LMs with logistic functions.

9.5.1 Fast evaluation with skipping NN LMs

This section proposes an improvement over the memoization of softmax normalization constants[55] described in [Zamora-Martínez *et al.* 2009a; Zamora-Martínez 2012]. Two different alternatives exist depending on what to do when a normalization constant has not been pre-computed: to compute it on-the-fly[56] or to construct what has been coined a "Fall Back NN LMs" as follows:

- During training:
  - train a NN LM for bigrams, trigrams,... up to n-grams;
  - pre-compute the softmax normalization constant for most probably expected n-grams for the higher order model and, for the rest of model, the most probably expected n-grams used when backing-off. Note that, when backing-off to bigrams, it is easy to pre-compute the entire set of constants, which is a table of the same size as the short list.
- and during recognition (as illustrated in Figure 116):
  - look for the $n-1$ gram input in the table of pre-computed normalization constants. If this value is not found, try on the lower order until the value is found[57] as illustrated in Figure 116;
  - compute the forward step of the corresponding MLP up to the last hidden layer;
  - compute only the activation of output neurons associated to the subset of words we are looking for. Note that this subset is usually a small percentage (otherwise, this approach would not be interesting); and finally

55 The sum of activation units of the output layer, after applying exponentiation.

56 A rigorous interpretation of the term *memoization* in the spirit of [Michie 1968] would only match the on-the-fly approach.

57 Convergence is guaranteed because bigrams contain all possible values.

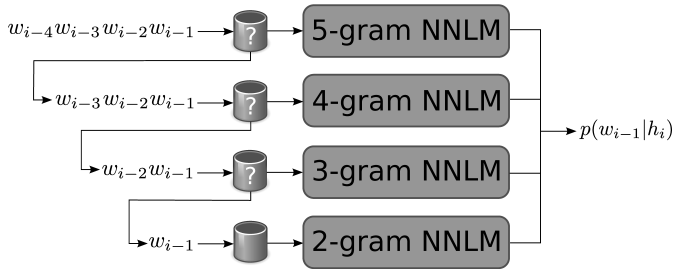


Figure 116: Scheme of a 5-gram Fall Back NN LM composed by four nnlms with precomputation of the normalization constants for the most probable inputs. If the required softmax normalization constant for the query of input context $w_{i-4}w_{i-3}w_{i-2}w_{i-1}$ is found in the table of constants for the 5-gram NN LM, the probability $p(w_i \mid h_i) = p(w_i \mid w_{i-4}w_{i-3}w_{i-2}w_{i-1})$ is calculated. Otherwise, the query is delegated to the 4-gram NN LM and so on. Note that the normalization constant is always found for the bigram NN LM.

– obtain the probability using the softmax formula using the pre-computed normalization constant.

A clear improvement over this basic idea is not to store normalization constants that would essentially lead to the same LM scores as if the system falls back to lower order NN LMs. We have not had, for the moment, the opportunity of measuring the effect of this novel proposal.

skipping NN LMs

Another proposed improvement over the previous scheme is called "fall back skipping" NN LM.[58] The idea resembles the skipping n-grams described in [Goodman 2001; Section 4] and in Section 6.10.2. The proposed approach, illustrated in Figure 117, consists in:

- using a unique MLP for all different orders;
- perform a back-off in different ways, similar to the "back-off graphs" from [Bilmes and Kirchhoff 2003]. The technique to remove a word in the past history consists of replacing this word by a special symbol we have denoted <NONE>. Note that, when using factored LMs, the observations made here relating "input position" should be generalized to "a factor of an input position".

The training procedure is as follows:

- include an special lexical item <NONE> to model the absence of input in a given position;
- when training the network by means of backpropagation, randomly replace some inputs by the <NONE> symbol in those places where back-off would remove them during recognition;
- pre-compute and store softmax normalization constants for more probably observed n-grams as well as for the same when some

58 Let us remark that, despite similar nomenclatura, skipping NN LMs, as described here, are not related with the continuous skip-ngram approach proposed in [Mikolov *et al.* 2013] whose aim is to obtain word representations good at predicting nearby words.

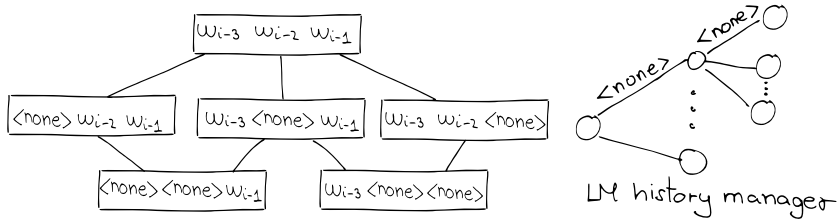


Figure 117: Scheme of skipping NN LMs. On the left, a lattice defined by the replacement of some lexical items by the <NONE> special symbol in the past history of a 4-gram skipping NN LM. Right: A LM history manager in the form of a trie may store the normalization constants associated to past histories including histories comprising <NONE> symbols. A NN LM as depicted in Figure 84 can be used to evaluate those combinations of the lattice whose softmax normalization constants are found in the LM history manager.

positions are replaced by <NONE> and, specially, for all combinations corresponding to bigrams.

Finally, during recognition:

1. given the input $n-1$ n-gram, look for the constants associated to different combinations of <NONE> from to to bottom in the lattice of Figure 117 (left). At least one constant is found since the bigram emulation is completely stored;
2. for each constant found, compute the network up to the last hidden layer. Observe that the first hidden layer can be computed in an incremental way if the similarities between different patterns is taken into account;
3. compute the LM scores using the softmax function; and finally
4. combine the different LM scores, combination weights can be estimated during training using a validation dataset.

Relating the reduction of the number of normalization constants, we propose the following simple greedy approach. It is based on inserting constants from lower (many <NONE> values) to higher order, avoiding the insertion of constants which do not produce a difference w.r.t. its absence: 1) find a topological order of the lattice of Figure 117 (left) from the bottom to the top, and 2) apply the following procedure for each mask and, given this mask, for each n-gram sequence from the training set:

1. compute the output of the MLP for the input n-gram sequence after applying the <NONE> mask;
2. measure the difference between the output of the model w.r.t. the result obtained without this pre-computed constant. The entropy can be used as in other LM pruning techniques [Stolcke 2000b];
3. store the constant if the discrepance between using and not using this pre-computed constant is significantly different.

This process seems quite costly, but is not performed during recognition and many steps can be parallelized or it could be simply avoided. Note that, although this procedure can also be applied to the original

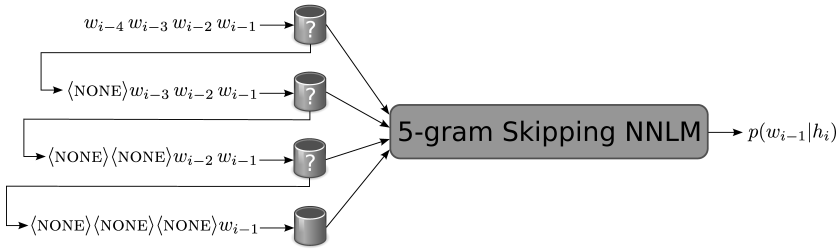


Figure 118: Scheme of a 5-gram Fall Back Skipping Neural Network Language Model composed by one 5-gram skipping NN LM and four precomputed tables of constants. If the softmax normalization constant is found in the 5-gram table for the input $w_{i-4}w_{i-3}w_{i-2}w_{i-1}$, the query is processed and the probability $p(w_i \mid h_i) = p(w_i \mid w_{i-4}w_{i-3}w_{i-2}w_{i-1})$ is computed. If not, the query is delegated to the same NN LM but with the input <NONE> $w_{i-3}w_{i-2}w_{i-1}$ at the 4-gram table, and so on.

fast evaluation approach from [Zamora-Martínez *et al.* 2009a; Zamora-Martínez 2012], what makes skipping NN LMs more suitable for this pruning technique is that the same MLP is being used in all the cases. This brings the possibility of incremental computing the different input combinations and the possibility of measuring the influence of certain inputs by computing the error gradient, or by directly comparing the activations of the last hidden layer.

emulate lower order n-grams

Some preliminary experimental work has been performed in a joint work with Pako Zamora, Adrián Palacions and María José Castro [Palacios-Corella *et al.* 2014]. These preliminary experiments, summarized in Section 13.2, leads to the conclusion that, at least for the task and corpora used in the work, the advantage of skipping NN LMs is basically restricted to the emulation of lower order n-grams. In this way, the original Fall Back NN LMs suffices, although it is greatly *simplified* since it is now based on just one MLP. Compare the scheme shown in Figure 118 with that of Figure 116.

Let us finally observe that, when we have been talking about using just one MLP during all the previous discussion, this MLP can be replaced by an ensemble of MLPs which, in turn, can be distilled into just one model afterwards using the ideas of [Hinton *et al.* 2014].

9.5.2 Estimating the normalization constant by an auxiliary LM

A novel way to avoid the estimation of the softmax normalization constant is now proposed. Contrarily to the previous one, it does not need to precompute any constant and does not require to fall back to lower order models either. As with the previous approach, it only makes sense when LM probabilities have to be computed for only a tiny fraction of the vocabulary.[59] It is based on estimating this normalization constant with the help of an auxiliary LM.

The usual way to compute the probability $o_i(h) = p(w_i|h)$ for word w_i, given the past history h, is obtained from the entire set of acti-

59 Otherwise, computing the exact constant does not imply an extra cost.

vations of the output layer a_i after applying the softmax function[60] (Equation 9.8 replicated here):

$$o_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)}$$

It is obvious, from the above equation, that it suffices to know a positive output $o_i > 0$ and the corresponding activation a_i in order to solve for the normalization constant $Z(h)$:

$$Z(h) = \sum_j \exp(a_j) = \frac{\exp(a_i)}{o_i} \tag{9.1}$$

The novel idea consist in using the probability $\tilde{o}_i = \tilde{p}(w_i|h)$, obtained with an auxiliary LM $\tilde{p}$, instead of the true one:

$$\tilde{Z}(h) = \frac{\exp(a_i)}{\tilde{o}_i} = Z(h)\frac{o_i}{\tilde{o}_i} \tag{9.2}$$

We can expect that $\tilde{Z}(h) \simeq Z(h)$ as far as $\tilde{o}_i(h) \simeq o_i(h)$ when assuming that similitude is referred to a low relative error. Indeed, LM probabilities by using this technique are scaled by the ratio[61] $\tilde{Z}(h)/Z(h)$. However, this approach is not exempt of drawbacks:

- suffers numerical stability issues when $\tilde{o}_i(h)$ is close to zero;
- since the auxiliary LM is, by definition, different from the actual one we are trying to approximate, we have no a priori way to bound the incurred error.

Since this trick can be applied, in principle, to any of the words of the (shortlist) vocabulary, we can try to improve the estimation by using several words at the same time. Moreover, this can also bring us a confidence interval on the value of the ratio $\tilde{Z}(h)/Z(h)$. We can therefore consider the following degrees of freedom:

- how many words and which words are selected to estimate Z;
- which rationale can be used to select a reasonable auxiliary LM or even several ones;
- how to combine the information from these words, namely: their activations in the true NN LM and the probabilities assigned to them, for the same LM context, in the auxiliary LM (or LMs, since we can have several ones).

The chosen lexicon subset should preferably satisfy the following conditions:

- the probability must be greater than zero and as high as possible;
- the more words the better, but too much means an increased computational cost. In the limit, the approach is unnecessary since all the activation values required to compute the softmax constant have been computed.

60 Dependence on the history h is not depicted in $o_i(h)$ and $a_i(h)$ for succinctness, not a problem since we are using always a sole LM history.

61 Which is $o_i(h)/\tilde{o}_i(h)$. Moreover, leads to an additive error when expressed in log probabilities.

Relating the choice of an auxiliary LM, the most obvious choice is a count-based n-gram. Indeed, the following experimentation is based on using the same LM that is ordinarily combined with MLP outputs in NN LM experiments. Nevertheless, we believe that the most convenient technique to obtain $\tilde{o}_i$ consists in using *the same MLP* provided with an auxiliary softmax output with a much smaller output size (in the limit case when this size is two, a logistic output would be equivalent). To this end, the original shortlist vocabulary is further reduced to the set of words used by the normalization estimation described before plus an output neuron used to group the probability mass of all words out of this selected subset.

Loss minimization approach

Given a LM context (or past history) h, let us see how Z can be estimated when using the information obtained from several words. Assuming that we have selected N words from the NN LM output (usually, the shortlist vocabulary) and that we have obtained their respective probabilities using the auxiliary LM: $\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_N$. We will assume that $\tilde{p}_i > 0 \forall 1 \leqslant i \leqslant N$ removing those words whose probability do not satisfy this constraint. At the same time, we compute the activation values (before the softmax) of the same words, and for the same LM history h, using the NN LM: $a_1, a_2, \ldots, a_N$. This only requires to compute the neural network up to the last hidden layer and the output neurons of the selected words. We seek to obtain an estimate $\tilde{Z}$ of the normalization constant Z which minimizes the overall loss incurred in this set of N words. When this loss can be expressed as the sum of the losses associated to each sample we have the following expression:

$$\tilde{Z}(h) = \underset{Z'}{\operatorname{argmin}} \sum_{i=1}^{N} L(\tilde{p}_i, \exp(a_i)/Z') \tag{9.3}$$

Several loss functions matching this form can be envisaged at this point. Since we are working with probabilities, it would seem natural to consider the cross entropy (Equation 6.1 replicated here):

$$H(p, q) = -\sum_{x} p(x) \log q(x) \tag{9.4}$$

The problem is that we are not dealing with a complete distribution but with a tiny fraction of samples.[62] A more reasonable loss criterion is the mean square error:

$$\tilde{Z}(h) = \underset{Z'}{\operatorname{argmin}} \sum_{i=1}^{N} \left(\tilde{p}_i - \frac{\exp(a_i)}{Z'} \right)^2 \tag{9.5}$$

62 Moreover, additional restrictions such as assuring that estimated probabilities lies between 0 and 1 should be also explicitly included. Without them, the minimization would lead to $\operatorname{argmin}_{Z'} / (\sum_{i=1}^{N} -\tilde{p}_i a_i) + Z'(\sum_{i=1}^{N} \tilde{p}_i))$. The first part can be removed since it does not depend on Z', the second means that Z' should be approached arbitrarily to zero.

The minimum can be obtained by setting the gradient to zero, which leads to:

$$\tilde{Z}(h) = \frac{\sum_{i=1}^{N} \exp(a_i)^2}{\sum_{i=1}^{N} \exp(a_i)\tilde{p}_i} \tag{9.6}$$

Aggregation approach

An alternative way to obtain an estimate consist in using the former Equation 9.2, which took just one output value into account, and to use the values $a_1, a_2, \ldots, a_N$ to obtain an estimate of a category that includes these words. An experimental comparison of both estimation approaches is described in Section 13.3

9.5.3 Pre-computing the input layer and NN LMs specialized for small vocabulary

NN LMs have been successfully applied in lexicon-free approaches for HTR tasks [Zamora-Martínez *et al.* 2010b]. In this section we propose an enhancement to reduce the computational cost of computing the NN LM probabilities.

Although the proposed technique is general and can be applied to any type of NN LM based on MLPs, it only speeds up the computation of the first hidden layer, so the effect on the global cost is limited by Amdahl's law[63] and, consequently, the expected gains are significantly higher in the proposed small vocabulary scenario. Perhaps, the relevance of this technique for the general case is increased when combined with other techniques suited to speed up the output layer. In general, the lexicon size is one of the main bottlenecks of NN LMs. But lexicon free approaches estimate sub-word based n-grams where the output size is the number of different sub-words, which is relatively small. For instance, the number of characters is comprised between 30 and 80, depending on case sensitivity. On the other side, the order of n-grams is raised, for lexicon free LMs, at values of 10 or even higher.

Amdahl's law

basic idea

When the lexicon size is very high, the computation bottleneck is placed at the output layer, which is no longer the case. But now the input layer contains more entries. The proposed approach speeds up this part. The idea consists of pre-computing the contribution on the activation units of the first hidden layer by each projection layer separately. This contribution is stored in a table where each possible lexical item at each input position is associated a vector of size the number of neurons of the first hidden layer.

As most approaches based on pre-computing and memorizing some values, there is a trade-off between space consumption and cpu time. This technique requires a matrix of size $|\Omega| \times (n-1) \times |h_1|$, being $|\Omega|$ the vocabulary size, n the n-gram order and $|h_1|$ the size of the first hidden layer. Note that this table can be reduced as far as a value not found during recognition can be computed on-the-fly. For instance, when the vocabulary size is very large, we can pre-compute and store only the most frequent lexical items.

63 This law models the expected improvement of the overall system when only some of its parts are improved. See http://en.wikipedia.org/wiki/Amdahl%27s_law

numerical example

The following example is a realistic setting for a HTR task using the IAMdb corpus as described in Chapter 15, i.e. a character-based NN LM 11-gram and a set of 78 characters. A MLP topology with a sole hidden layer of 512 units with a projection layer of 32 units.[64] This MLP topology requires $(79+1)\times 32+10\times(32+1)\times 512+(512+1)\times 78 \approx 211.5$K parameters occupying 826 Kbytes, approximately, using single precision values. We have included a special input context cue but no <EOS> context cue at the output. The projection layer weights are only counted once because they are usually shared.

The costs of a pre-computed table to speed up the computation of the hidden layer would require $79\times 10\times 512 \approx 404$K parameters occupying 1.58 Mbytes. This is high compared to the size of the net, but not so much compared to the size of other LMs or the size of the pre-computed constants for a word based NN LM.

Let us now compare the cost of computing the first hidden layer with and without the proposed technique:

- the traditional approach, replacing the projection layer with a table look-up, would require around $10\times 32\times 512+512\times 78$ additions and multiplications and $512+78$ exponentiations, around 410K floating point operations assuming that additions and multiplications have the same cost and that exponentiation is around five times more costly. Note that the cost of the input layer constitutes around 80 percent of the global cost;
- using the pre-computed data, we would only need 10×512 additions for the hidden layer,[65] and the same 512×78 additions and multiplications and $512+78$ exponentiations for the rest of the network, leading a total of 48.5K floating point operations, which means an 8.4 times faster or, alternatively, to use only the 12 percent of the original time.

an order of magnitude faster

This approach require to modify the forward step of MLPs to solve an ad hoc problem, which seems a drawback. Another problem is the space required to store the partial sums of the first hidden layer. This space is increased a lot when using larger vocabulary sizes. It is better, in this case, to store only most probable words.[66] We believe it is worth implementing this approach because the required modifications in our neural network toolkit are not very complex.[67] We have recently known, much time after being designed this approach (and when this part of the report was already written) about the existence of [Devlin *et al.* 2014; Section 2.4] where the similar idea is proposed in general for NN LMs without any reference to the particular topology where this improvement would be more relevant.[68]

64 The size of the projection layer is irrelevant in the pre-computed approach. That is why, we would choose other size or even to completely remove this layer. The value can affect the relative cost of the computation when the technique is not applied and, hence, the expected gain.

65 The vector is initialized with the bias

66 Let us remember Zipf's law.

67 This improvement has been proposed during the development of [Zamora-Martínez *et al.* 2009a] more than five years ago but, as with an uncountable number of ideas, we are a too tiny research group with plenty of other occupations.

68 Indeed, the more the softmax cost is avoided the more the current proposal becomes relevant. This section was written not only before being aware of [Devlin *et al.* 2014]

9.5.4 "LM look-ahead" NN LMs

LM look-ahead (LMLA) NN LMs are a new type of LM proposed in this work to include a feature very useful for decoding:

> *MLPs can contain several independent softmax output layers. Let us exploit this capability to estimate the probability associated to each branch in a network*[69] *lexicon decoder.*

In this way, the probability of a word given its previous history $p(w|h)$ is no longer estimated by a sole neuron from a softmax layer but it is rather computed as the composition of the different branches required to reach this word in the lexicon tree, mimicking the same sequence of decisions that a tree lexicon decoder would perform, as illustrated in Figure 119. This approach has, in our opinion, some advantages:

- instead of computing a large softmax layer, we only have to deal with tiny layers. This removes the problems of computing costly softmax normalization constants;
- these values can be computed on-the-fly as states from a lexicon decoder become activated. More importantly, these values constitute *exact*[70] LMLA values for free, which is the reason we have coined them "LMLA NN LMs".

removing one output in softmax layers

It is possible to take profit of this particular MLP topology to propose a simple improvement[71] which consists in removing one of the output neurons of each softmax group of neurons in the output layer. There is little gain in the general case but, now, LMLA NN LMs have a large number of small softmax groups. Since the softmax output is invariant to any offset equally applied over all output activations:

$$\frac{\exp(a_i + c)}{\sum_j \exp(a_j + c)} = \frac{\exp(a_i)\exp(c)}{\left(\sum_j \exp(a_j)\right)\exp(c)} = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \tag{9.7}$$

it is possible to subtract the weights of an arbitrarily chosen one output neuron to the same weights in all the output layer. Its activation becomes zero and its weights can be removed. It is clear that branches with two children are tackled with a logistic activation function (assuming that $a_2 = 0$ and dividing by $\exp(a_1)$):

$$o_1 = \frac{\exp(a_1)}{\exp(a_1) + \exp(a_2)} = \frac{\exp(a_1)/}{(\exp(a_1) + \exp(1))/\exp(a_1)} = \frac{1}{1 + \exp(-a_1)} \tag{9.8}$$

In spite of those qualities, LMLA NN LMs have some drawbacks:

- the internal structure of the LM is dependent on the particular lexicon transcription, which makes different parts of decoders more interdependent;

but also much before 2014 (as stated in previous footnote) and, also, before the other sections proposing ideas for avoiding the cost of computing the softmax normalization.

69 We will only discuss the case of tree lexicon decoders. Nevertheless, this idea can be extended to more general lexicon network decoders as those described in Section 10.10.

70 Meaning that it is not an approximation, it is not related to the LM accuracy.

71 Although is a naive idea, we have not found it described elsewhere.

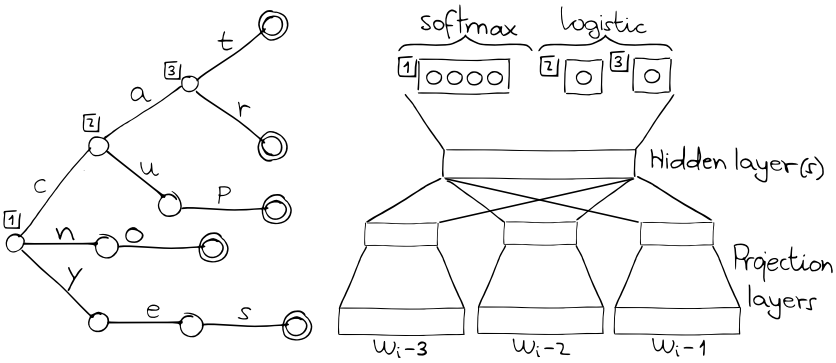


Figure 119: LMLA NN LMs. On the left, a tree lexicon for the toy vocabulary $\Omega = \{$ car, cat, cup, no, yes $\}$ with three nodes with a branching factor greater than one. They are depicted by a small number within a square. On the right: a LMLA NN LM with several output layes: a softmax layer for branching factors greater than 1 and a logistic unit for each branch with two branches (associated to the labeled nodes of left figure).

- it requires some special capabilities of the neural network toolkit, namely, the capability of specifying several output softmax layers and how to train them properly: When training a conventional NN LM with a given n-gram patter, the $n-1$ word prefix is used as input pattern and the last word suffix is used to select which output is set to 1. However, in a LMLA NN LM, this would lead to several softmax groups to be unused. We can envisage two different approaches to tackle this issue. One the one side, to be able to specify that certain output layers are not used and that no error has to be backpropagated. On the other side, to create a soft-target where the output probability of all the words of the lexicon is estimated and provided to the training toolkit as desired output. These soft-values can be estimated from a previously trained NN LM ensemble. The values of the softmax outputs associated to each lexicon tree branch can be computed from this soft target in the same way as LMLA probabilities (as described in Section 10.6.7). This approach can benefit, as in other cases described in this chapter, from the use of higher temperatures to improve the training [Hinton *et al.* 2014];
- the LM interface and the decoders using this type of LM have to be adapted to take profit of LMLA, as described in Section 10.6.7.

Relating prior art and similar ideas, this approach is, to the best of our knowledge, totally novel. Nevertheless, some resemblances can be found with the idea of structured output [Le *et al.* 2011] where just a hierarchy is used without any relationship with the tree lexicon used for decoding.

A more related work using structured output to estimate a LM in a hierarchical way as reported, for instance, in [Morin and Bengio 2005]. Their main aim was to speed-up NN LMs and they propose

to structure the lexicon in a hierarchical way, but they use binary trees constructed by means of hierarchical clustering based on prior semantic knowledge. Since the idea of using a hierarchical output is not new, we can conclude that the distinguishing feature of our LMLA NN LMs proposal is the capability of computing LM look-ahead and to compute them on demand in a quite tied implementation co-design between LM and decoder where the last hidden layer is memorized and outputs are computed on demand.

Let us remark that there exists other alternatives to obtain a cheap LMLA by means of NN LMs. It suffices to simulate a word based LM by using a sub-word based LM as described in Section 6.9.2. A mixed approach between both LMLA NN LMs and sub-word based NN LMs consists of using a lexicon free LM trained with sub-word models automatically extracted from the trie lexicon branches and to place the past history (like word based NN LMs) as an additional input to the network.

mixed approach

This idea has two obvious drawbacks. On the one side, several forward steps are required when only one sufficed in the trie based approach. On the other side, the model does not guarantee the probability mass to sum up to one, as pointed out in Section 6.9.2. Another way to benefit from the combination of a word based NN LM and the lexicon free version is to deal with OOVs as explained in Section 6.8.

9.6 ASSISTED TRANSCRIPTION LM REPRESENTATION

Chapter 6 introduced the more theoretical aspects of language modeling. Besides the description of many common types of LM, that chapter also introduced an approach related to decoding, namely: the use of language modeling to deal with some features that other works tackle by modifying decoders in some special or ad hoc ways. Some LM wrappers were described to this end. Now, the implementation of one of these features will be addressed.

An example to illustrate the interest of dealing with some specific decoding problems by means of language modeling is the case of interactive assisted transcriptions, described in more detail Section 3.2.2. Although there exists several methods to deal with interactive systems, a common one consists in correcting the sequence following a left-to-right interaction protocol so that the prefix validated by the user grows in length until covering the entire sequence.

constrain validated prefix

We have only been able to found in the literature very imprecise descriptions of the LM used by these systems. [Rodríguez *et al.* 2007; Toselli *et al.* 2007a] describe the use of a special language model called suffix LM or prefix constrained LM (see Figure 120) which is described quite succinctly as:

> *the concatenation of a linear model which strictly accounts for the successive words and the suffix language model*

whose probabilities are obtained by modifying particular formulas for count based n-grams.

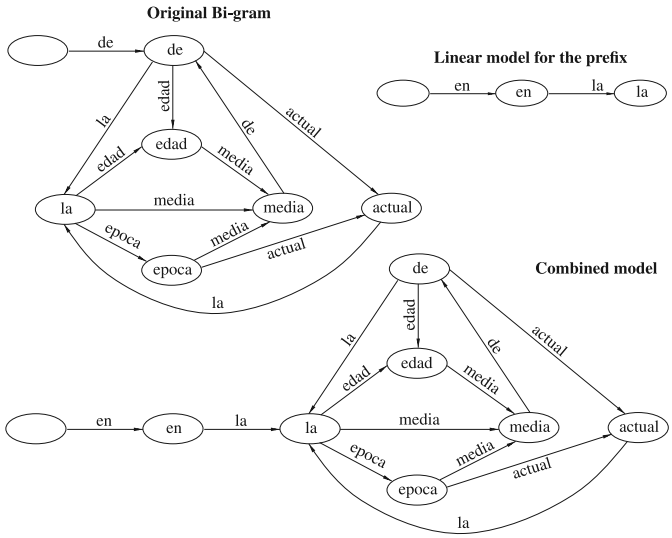


Figure 120: Figure reprinted from [Toselli *et al.* 2010; Fig. 3] showing how to build a special language model useful for assisted transcription of text images. This LM can be seen as the "concatenation" of a linear model which strictly accounts for the successive words in a consolidated prefix p and a "suffix language model".

Unfortunately, we have not found any implementation detail in these works in order to properly quantify or compare the computational effort and memory footprint of this approach.

LM agnostic technique

We propose, in this work, a more flexible and general approach by means of a LM wrapper to use any LM with consolidated prefixes by combining the following models

- the general LM which has been used in a non interactive transcription for the same task, which is not constrained to be a count based n-gram;
- a model which accounts for the language of sequences sharing the consolidated prefix. This language can be easily modeled with a finite state automaton which recognizes a prefix followed by a simple loop. Instead of using a unigram loop, we can make the model non-probabilistic. In this case, the combination is not performed as described in Section 6.4 but, rather, as a mechanism to constraint the first LM with the second.

The main advantages of this approach are an efficient implementation in a dynamic decoder and the fact that it can work with any LM without requiring specialized formulas. This proposal has been empirically validated in [Castro *et al.* 2011].

The current implementation is based on a simple dictionary (hash table) which associates an hypothesis word length and the LM identifier of the original LM (the one which would have been used without receiving any feedback) with a new LM identifier. When a new LM transition is demanded, we simply compute a new tuple by increasing the word counter if this value is lower than the consolidated prefix, in other case this value is ignored and only the underlying LM is ap-

plied. However, if the word count is lower than the consolidated prefix length, only transitions associated to the corresponding word are allowed, the rest are simply discarded. Note that this is not a LM combination in a probabilistic setting but rather a constraint applied on the original LM. This approach is similar, although much more simplistic and restricted, than the case of imperfect transcriptions described in Section 6.9.3.

9.7 SUMMARY AND SOME CONCLUSIONS

This chapter complements Chapter 6, which was more theoretically focused. Now, we have mainly discussed LM interfaces, representations as well as LM look up algorithms and their implementations. In the first case, we have reviewed the most typical LM interfaces, paying special attention to the automata based representation. We have also described our LM interfaces describing some additional features: the inclusion of pruning criteria and some methods for simultaneously querying a large set of LM look-ups. The capability of jointly specifying a large number of queries (an approach that we have coined *bunch mode*), together with pruning inside the LM, allows certain LM representations to speed-up the global process. This has also been co-designed with some decoders which may benefit from these novel capabilities.

LM interface extensions such as bunch mode

Relating other contributions at the LM interface level, we have also discussed the benefits of a feedback mechanism between the LM and the decoder allowing the segment estimator to restrict which lexicon decoders would be more suitable at a given moment.

novel feedback to select decoders

Moreover, an extension to deal with RTNs has been presented. A worth mentioning particularity of the RTN formalism used in this part is the possibility of using reference arcs with several exit points, which we have metaphorically compared with forks. The usefulness of this feature, which we have not found elsewhere, is motivated by their potential to model the lexicon tree approach making the reuse of tree decoders in the dynamical setting[72] just a particular case.

RTN extensions

The second part of this chapter is devoted to LM representations. Again, we have reviewed the most common techniques found in the literature. We have described the representation we have implemented in this work to represent count based n-grams converted into automata. The idea is very simple and naive. States are sorted by their fan-out, making it possible to easily distinguish between different search operating regimes: using a dense vector, a linear search, a binary search or an interpolation search. These different techniques can be further improved when dealing with bunch mode and, even for single LM queries, from the appropriate LM query policies.

fan-out based automata representation

NN LMs have also had a place in this chapter. Although they are not the main focus of our thesis, we have used them in some experimental work and some improvements have been proposed in this chapter. A review of common techniques to speed up the evaluation of this type

some NN LMs related contributions

72 See the introduction of Chapter 10, the section titled "Motivation: Dynamic vs Static" explains the difference between both types of decoder.

of LM is provided together with some ideas which could be used at this point: 1) an improvement to the idea of [Zamora-Martínez *et al.* 2009a; Zamora-Martínez 2012] consisting in memoizing normalization constants has lead to the definition of *skipping NN LMs*, which are not to be confused with other works using nearly the same terms. An empirical validation of these models, in a joint works with other authors, has shown that they are very well suited for emulating lower order n-grams. More experimentation would be required to further assess the usefulness of the more general setting described here. 2) A novel technique to avoid the computation of the softmax normalization constant by estimating it *with the aid of an auxiliary LM* (and even better, an auxiliary softmax output layer in the same NN LM) is described and accompanied with some preliminary experimental measurements. 3) a way of improving the estimation of logistic outputs from soft-targets obtained from a previous (ensemble of) softmax-based NN LMs at higher temperature. 4) a way to speed up the input layer (which perfectly fits lexicon-free approaches) and, to finish this neural network related part 5) a novel type of LM that we have coined "LMLA NN LM" because it allows the direct computation of exact LMLA scores "for free".

novel LMLA-NN LMs

The clever reader may wonder why to worry about these techniques when other proposals to reduce the cost already exists, namely: shortlist approach, structured output layers and, perhaps the most clear example, training the MLP by including a regularization term so as to reduce the variance of the softmax normalization constant. To this respect, we have to remark that proposing new techniques makes sense since these techniques have not been consolidated as *definitive* solutions yet. Moreover, some of these techniques could be used together.

Finally, the last section describes a "LM agnostic" approach to deal with interactive transcription tasks. This is related to the ideas of Section 6.9.3.

As future work, we plan to further improve the LM interface by including new features as novel decoders demand them. One example worth investigating is the possibility of providing iterators to get the sequence of transitions ordered by score. This could be combined with another sequence also ordered by score generated by the lexicon estimator module. It is possible to combine them on demand using a pruning technique inspired by the ideas of [Patel 1995; 1997] briefly explained in Section 10.6.9.

Another future work project is to take advantage of the bunch mode at the LM interface to implement a distributed LM representation where a large LM is split into different computers [Brants *et al.* 2007]. The capability of managing large sets of LM and the division of the automata based LM by numerical properties of state identifier queries makes it easier to efficiently distribute queries between different computers and to merge the results afterwards (see Section 12.1.1).

distributed LMs

10 SEGMENT AND LEXICON ESTIMATORS

PREMATURE OPTIMIZATION IS THE ROOT OF ALL EVIL.

Knuth (doubtfully citing Hoare)

CONTENTS

10.1 INTRODUCTION

THIS chapter is devoted to the design of decoders specialized in estimating the likelihood of generating a signal segment (a frame sequence or a span of the input DAG) given a basic unit from a predefined catalog (i.e. phones, graphemes, ...). What is considered a basic unit is not necessarily inherent of the task: isolated speech recognizers with small vocabularies usually model whole words with HMMs, but it is also possible to represented words as the concatenation of sub-word units from the phonetic transcription. Other parts of the global decoder do not care which approach has been used. We will consider decoders estimating the conditional probability of observing either sub-word units or entire words from a lexicon, taking profit from the fact that words share sub-words models to reduce the computation effort and to improve the estimation of model parameters. The term "lexicon decoder" will be used to emphasize the fact they estimate words from a lexicon. Decoders presented in this chapter have in common some properties:

- both sub-word and lexicon decoders are part of other decoders (Chapter 12) or part of DAG generators (Chapter 11). The overall system uses LMs, keeping track of information to retrieve the output;
- in this way, lexicon decoders do not handle LMs directly. Nevertheless, LM look-ahead technique makes it necessary, excepting trivial cases, to refer to the LM;
- they do not keep track of information to retrieve the best path at sub-word level since this level of detail is usually not required, although this should be made possible in some cases. The manipulation of a (trellis) data structure is usually delegated to decoders which makes use of lexicon decoders and LMs. They are usually from a type known as "dynamic" (explained below).

This chapter is focused in HMMs,[1] but we will also study a decoder to estimate the probability of observing words in a DAG labeled with sub-word units (phonemes, graphemes,...), which can be used as a piece of a graph generator algorithm ofq Section 11.2.

Motivation: Dynamic vs Static

Most recognition systems use HMMs to model phonemes or graphemes. Likewise, they use n-grams or other finite state LMs. This combination or composition of models leads to a (huge) integrated finite state model. [Mohri and Riley 1998], and many others, propose to use a unique "static network" representation where several knowledge sources are represented in a uniform way by means of transducers. In this scenario, it is not required to distinguish decoders specialized in words or phonemes. What is the point of discussing them? They make sense in the dynamic search approach where they can be specialized to take profit of particular properties.

1 Segment units other than HMMs, reviewed in Chapters 4 and 7, usually deserve a special treatment. Unless otherwise stated, our scope are HMMs.

static vs dynamic

Static and dynamic approaches are usually considered opposed. This nomenclature may seem misleading since the static approach can use on-demand composition and determinization algorithms. In this way, only the required part of the search space needs to be constructed during the recognition stage. This approach has pros and cons. On the one side, a compact static representation can be efficiently compiled and stored beforehand. This is particularly important because their construction can be very time and space consuming. On the other side, the use of on-demand manipulation makes it possible to take into account dynamic LM capabilities. Moreover, dynamic decoders are not the only approach which can take profit of specialized models, at least from the representation point of view: Specialized representations can also be used in the static network as long as they offer the proper interface, as discussed in Section 9.1.2.

Static decoders have several advantages. For instance, weighted determinization does not only reduce the search space, but also performs a weight (probability) look-ahead. This improves, in average, the performance of pruning techniques.

The dynamic search space approach has several similarities with the static counterpart. For instance, a similar interface can be used to interact with different types of LMs. Relating the differences between both approaches, lexicon hypotheses are usually computed, in dynamic decoders, after some observed (acoustic in ASR) evidence is obtained by means of a quite general lexicon decoder. In this way, LM look-ups are restricted to this subset. Contrarily, most on-the-fly composition algorithms used in the static approach require the traversal of all outgoing transitions the first time a state is reached, which seems wasting effort if only some transitions are likely to be used base on the evidence given by the signal, specially when the vocabulary size is huge.

Dynamic search approaches allow the use of very specialized faster decoders. These decoders work coordinately with other parts of the system in order to compose a whole decoder without explicitly constructing an integrated network. This interaction is achieved by exchanging tokens. The token-based paradigm [Young *et al.* 1989] is also discussed in Section 10.2.3.

more flexibility

Our motivation for choosing the dynamic approach instead of the static one is not due to their relative performances. We have prioritized its flexibility for using different models types and for the possibility of combining them in different ways at different levels. For example, although it is possible to use NN LMs in the static approach, these models would only make sense in the particular case of on-demand composition. We believe that they are perhaps more suited for dynamic search decoders. Also, the transducer composition approach is based on the assumption that all information sources can be represented by means of transducers, which is not always the case: it is possible to use context free language models.[2] Another example of non-regular models are some generative segment models such as the combination of generative and discriminative segment models proposed in Section 7.7. Another feature which seems more easily described in the dynamic

2 To this respect, there exists approximation techniques to represent these models in terms of transducers [Cortes and Mohri 2000; Nederhof 2000].

search approach is the use of over-segmentation information at some particular levels (e.g. at the word level in offline HTR), although we cannot a priori discard the possibility of good modeling techniques of these features in the static paradigm. This chapter deals, therefore, with the design and implementation of some decoders specialized in one of the most time consuming parts of dynamic search recognizers, although other settings (e.g. asynchronous stack decoding techniques) can also benefit from decoders reviewed here.

10.2 SEGMENT ESTIMATOR INTERFACE

Let us first start by discussing the interface which could be required to use segment and lexicon estimators by other parts of decoders. This does not seem very complex *at a first glance*. Indeed, a first naive proposal which comes to our minds is a method associated to the segment model which receives a sequence (a vector of frames) and returns the corresponding score (i.e. logarithm of a scaled likelihood):

```
Score estimate_score(Segment s)
```

But this interface is clearly insufficient because there are other issues that should also be taken into account, namely:

- incremental computation of hypotheses;
- estimation of several units/words by the same decoder to reduce the overall computation effort. Chances are that many words share common sub-words;
- hypotheses at the sub-word and lexicon levels are associated to other information which is managed by the global decoder. A common way to provide an abstraction to allow the lexicon decoder to manipulate this information in a homogeneous and general way is the "token passing" paradigm discussed below, the basic idea is that tokens are abstract entities which are passed between hypotheses and combined by means of a generic interface decoupling the bookkeeping and management of the output data structures;
- across-word context dependent decoding requires for emitted hypotheses to contain information about the right context of the last sub-word. This information is properly managed by other parts of the decoder (to obtain context-dependent word-graphs, for instance) and is finally used in the input of lexicon decoders to properly start hypotheses so that the initial left-context is compatible;
- maintain only a set of active states and apply pruning techniques to avoid an uncontrolled growth of this set at the expense of losing optimality.[3]

3 Several studies have empirically shown that this lost is limited in practice and that there exists a trade-off with a typical diminishing returns curve where huge amounts of computational effort are required to obtain negligible improvements.

10.2.1 Incremental computation

Usually, when computing the likelihood of generating a sequence of frames $x_1, x_2, \ldots, x_n$, we also obtain *for free* the estimations associated to previous prefixes of the sequence: $x_1, \ldots, x_k$ for all $k \leqslant n$. This is the case of HMMs when using dynamic programming based Viterbi decoders. In those decoders, the score associated to each active HMM state is updated, for a given frame, with the emission probabilities estimated from this frame and with the scores from previous states.

Although this trellis is computed from left-to-right, nothing prevents us from computing these scores in a right-to-left way or even in more complex ways. Indeed, the steps of Viterbi and forward algorithms can be expressed as matrix multiplications. Vectors represent HMM state values associated to spans of the sequence and matrices combine transition scores with emission probabilities. The associativity of matrix multiplication makes it possible to compute the desired values in other ways different from left-to-right, as proposed in [Nielsen and Sand 2011] where the computation of the entire sequence can be parallelized, as illustrated in Figure 121. Unfortunately, this approach is more suitable for (and mainly restricted to) HMM topologies with a reduced number of states where the scores are computed for all states. This approach is not appropriate for some tasks with a huge number of states, as is the case of ASR and HTR. In those tasks, it is more reasonable to consider only those states with a non-negligible score (called "active states") and other parallelization techniques are suitable for these cases. Nevertheless, in our context, the former example from [Nielsen and Sand 2011] serves to illustrate the flexibility of incremental computation of segment emissions probabilities in ways other than left-to-right. Different ways of incremental computation, not necessarily limited to HMMs, are sketched in Figure 122.

Due to the practical relevance of the left-to-right approach, we will limit our interfaces to the left-to-right incremental way: segment estimators are modeled as objects with a method to add a frame to the sequence and other method to consult the computed estimation. The following methods should suffice to decode sequences:

NEW is the constructor to obtain a new estimator associated to an empty segment, it can receive information about which segment types are to be estimated since the same estimator can deal with several segment types;

UPDATE receives a new frame and updates the internal state of the segment estimator to include this frame at the end of the segment;

GET-VALUE receives a word identifier and returns the estimated score associated to this word.

Some segment models (for example, holistic MLPs combined with other generative models) do not allow incremental techniques and require the entire segment in order to start the computation. Fortunately, nothing prevents us from using the previous incremental interface with these models: **`update`** can just keep track of the partial input and delay the actual computation of the entire segment until it is required by **`get-value`**.

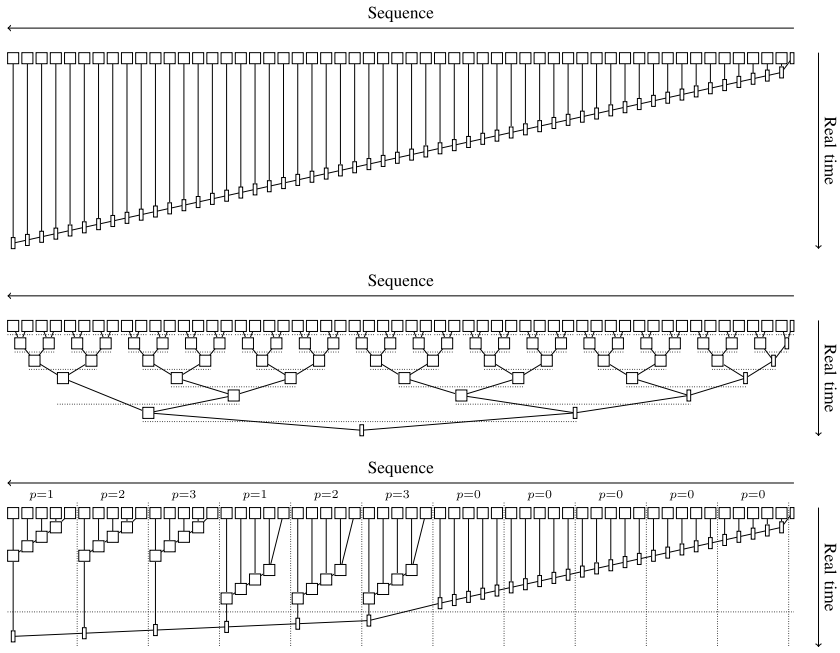


Figure 121: Images extracted from [Nielsen and Sand 2011; Figs. 2,3 and 4] illustrating the parallel reduction of forward algorithm: On the top, the traditional forward algorithm. In the middle, a balanced parallel reduction of the same algorithm. Finally, at the bottom, a more clever approach tailored for a specific number of processing units where each processor takes advantage of the sparse matrix properties of the traditional algorithm as much as possible. Rectangles represent matrices and vectors. Vectors represent HMM state values associated to spans of the sequence and matrices combine initial or transition scores with emission probabilities. Diagonal lines represent dependencies. Viterbi algorithm can be parallelized in a similar way.

10.2.2 Persistent and ephemeral operations

The vast majority of decoders are designed to work exclusively with frame sequences, but a sequence is a particular type of DAG where every edge represent a value and vertices represent the boundaries between consecutive values. One of the properties of sequences, when viewed as DAGs, is that vertices have only one predecessor (excepting the initial vertex) and only one successor (excepting the final vertex).

in place updating may not suffice

A decoder restricted to frame sequences only requires to update its internal state after processing each frame. However, vertices of more general DAGs may have more than one predecessor and more than one successor (Figure 123). That is why the same segment decoder state have to be used to perform several updates and the API proposed before does not suffice. Indeed, "Purely functional" or "persistent" are terms used to describe data structures that generate a modified version while the original data is accessible after applying the operation [Driscoll *et al.* 1989]. They can be used for several independent operations. Classical data structures which are updated in place are known as "ephemeral".

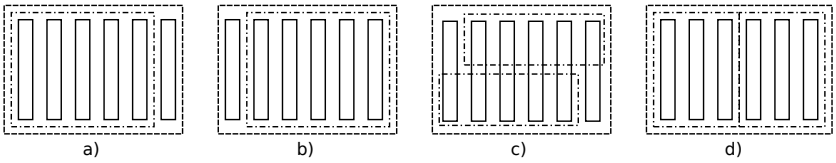


Figure 122: Different incremental segment estimation possibilities: (a) The estimation of a segment reuses computation from the estimation of its prefix, which is the typical case of Viterbi decoding with HMMs processing frames from left-to-right, (b) the estimation of a segment reuses information from its suffix, (c) both prefix and suffix are used when estimating a segment (bidirectional RNNs, when restricted to the computation of frame emissions and not segment likelihoods, constitute an example), and (d) segments are composed in a bottom-up fashion, as is the case of CFG productions $A \rightarrow BC$.

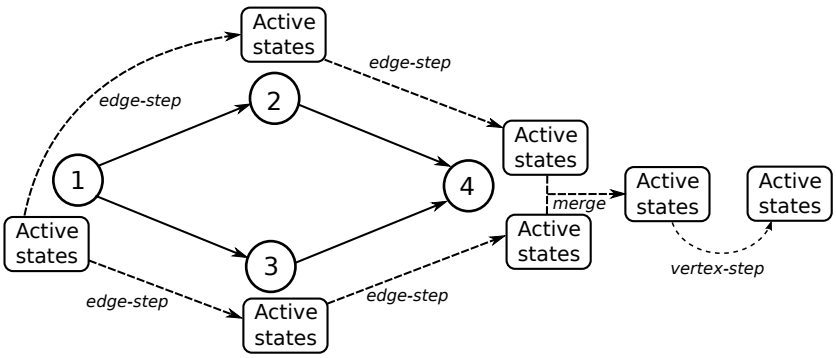


Figure 123: Example of the persistent API: active states of vertex 1 cannot be updated with edge $\langle 1,2 \rangle$ because they are also used for edge $\langle 1,3 \rangle$. Lexicon states have to be combined when several edges arrive to the same vertex, as illustrated in the merge operation associated to vertex 4. **`vertex-step`** has only been applied in vertex 4 and not in other vertices just to simplify the figure.

Persistent API

Estimators requiring the entire segment may use a functional list data structure to store incoming frames. Since the spatial and temporal cost of functional lists operations is constant per node [Okasaki 1995], the use of incremental updating is compatible with persistent estimators. It is important to distinguish between persistent and ephemeral operations in order to extend segment estimators from sequences to DAGs. Ephemeral updating suffices to describe decoder working on sequences, but we need to take into account other types of operation to work with general DAGs. Let us enumerate the methods exposed by the persistent versions of segment estimators:

NEW creates a new segment estimator. The segment estimator is usually associated to the initial vertex and requires some information relating the initial scores which are usually carefully scaled to make a pruning step compatible with other parts of the global decoder, this issue will be addressed below;

EDGE-STEP receives an edge and returns an updated segment estimator which includes this edge as the right part of the segments whose generative probability is estimating;

MERGE takes into account the information of several segment estimators which are associated to the same DAG vertex. This function combines the active states of every operand and is useful when a vertex DAG has a fan in or input degree greater than one;

VERTEX-STEP produce an updated segment estimator which has taken into account certain operations which do not consume sequence symbols as described in the example below;

GET-VALUE is the same as in the ephemeral case since this method does not usually modify the internal state.

Note that, due to the **`merge`** operation, those estimators can take into account the joint contribution of several segments (which corresponds to the computation of the semiring value associated to a given span of a DAG, as described in Chapter 5).

Example: error correcting DAG decoder

In order to better understand the rationale behind **`vertex-step`**, the following example requires a non-trivial **`vertex-step`** method: to estimate the probability of generating words from a phone DAG given a pronunciation lexicon and an error model (see Sections 10.4 and 10.9.5). The error model allows the following edition[4] operations:

- insertion of a phone;
- substitution of a phone by another one;
- deletion of a phone.

Let us see how to adapt the methods described before, details on the implementation of these methods are given in Section 10.4:

EDGE-STEP operation will take into account the correct and incorrect substitutions and also the deletions. Deletions are like a null transition in the input DAG because the set of active states is not used to traverse lexicon transitions. Instead, active states remain the same but scores are penalized by the cost of the deletion;

MERGE performs the usual work as if no edition operations exist;

VERTEX-STEP deals with the insertions. Since insertions do not consume edges, it would be a waste of time to apply these operations in the **`edge-step`** because these operations can be factorized and applied just once after all incoming edges has been merged. Regarding the serialization protocol of Section 8.2, this method can be associated to the `no_more_in_edges` message associated to the current vertex. Note that a `no_more_in_edges` message is mandatory for the initial vertex and, in this way, insertions can take place before observing any input edge, which would allow insertions at the beginning of the sentence.

4 Transposition of adjacent symbols is considered in some works, but not here.

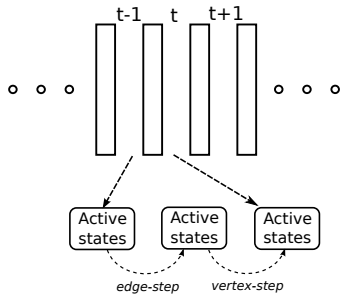


Figure 124: Ephemeral operations for frame sequences. Frames are edges and frame boundaries corresponds to vertices. a step of most classic Viterbi decoders corresponds to several persistent operations which are usually combined in a unique ad hoc specialized and efficient persistent updating. This simplification does not suffice to deal with general DAG input, but other ephemeral operations such as **`edge-Update`** allow the use of input DAGs.

Ephemeral API

Persistent operations can often be implemented more efficiently than the equivalent sequence of persistent. Also, this particular case suffices to deal with frame sequences. This is the reason why many works ignore the very existence of the persistent modality and only focus their attention on the ephemeral counterpart. The basic idea consist of joining several operations allowing an in-place updating of the involved data structures, as shown in Figure 124. Let us see how these ephemeral operations can be expressed as the composition of persistent ones which are obviously more general:

EDGE-UPDATE is a method on a segment estimator which receives and edge **`Y`** and another segment estimator **`X`**:

```
self = merge(self, X.edge-step(Y))
```

VERTEX-UPDATE simply performs the **`vertex-update`** method in an ephemeral way:

```
self = self.vertex-step(self)
```

UPDATE as we have seen before, is the method of choice when working with sequences. It can be described as the sequential composition of the former ephemeral methods:

```
edge-update(self,frame).vertex-update()
```

A note on confusion networks

Confusion networks (a.k.a. sausage nets) [Mangu *et al.* 2000] are a particular case of DAGs described in Section 3.1.15. They can be seen as a linear sequence of confusion sets where all the vertices are traversed by any path from the initial to the final vertex. They constitute another generalization of sequences which can take profit of persistent/destructive operations. It suffices to increase the API with a new version of **`update`** receiving the entire[5] confusion set, which is not a novelty since the **`edge`** method already considers the possibility of a mapping from edge symbols to scores as argument. Confusion networks allow the use of null transitions to describe sets of sentences of different length.

5 Since the "sausages" metaphor has been used to describe confusion networks, we can humoristically say that the system eats the sausage net *one sausage at a time*.

It is possible to combine the different error correction edition operations, described before, for a decoder using a confusion set in each step: the probability of the null transition in the confusion set would be combined with the probability of insertion. This lexicon network decoder could be used in a multi-stage system where a first stage produces a phone confusion network similarly to [Bertoldi *et al.* 2008].

Combining persistent and ephemeral operations

It is possible, even desirable in practice, to combine the ephemeral and the persistent APIs as well as to include some methods to combine more efficiently[6] some operations:

- the **`edge-step`** performed over the last edge message outgoing from a vertex can use the ephemeral modality;
- in most cases, an edge operation produces a value that has to be merged with the value previously present in the destination vertex. Both operations can be combined in the same process.

Note that not all lexicon estimator implementations can take profit of these combinations to speed up decoding, but they can be included without effort, using the basic operations, by means of inheritance.

10.2.3 LM-conditioned vs time-conditioned decoders and the Token Passing approach

As explained in the introduction, we are not interested in using segment estimators in isolation[7] but, rather, as components of other more global decoders able to recognize sequences making use of a language model. It is also clear that, when both the lexicon and the LM can be represented by means of transducers, the use of the static network expansion makes the very distinction of lexicon decoders even arbitrary.

One of the simplest ways to implement a one-pass decoder with a lexicon estimator is to mimic the transducer composition search space without the further improvements easily obtained in this approach: a lexicon decoder is associated to each LM state. In a dynamic setting, this can be implemented by creating lexicon decoder copies on demand. This approach is known as LM-conditioned since all the hypotheses inside each lexicon network copy share the same LM state. LM transitions have to be computed, in principle, only for tokens going from one lexicon network to another one (Figure 132b).

LM-conditioned

Observe that the fact of actually performing lexicon network copies can be considered an implementation detail. It is possible to use a sole lexicon decoder instance containing the hypotheses associated to different LM identifiers.[8] In this case, the LM identifier is used to distinguish different hypotheses associated to the same lexicon network state.[9] This is basically the idea of re-entrant decoder where the hypotheses emitted by the lexicon network are inserted again into it, as

6 This idea is exploited by functional programming languages supporting *unique types*. See `http://en.wikipedia.org/wiki/Uniqueness_type`.

7 Excepting for isolated word recognition tasks, obviously.

8 The idea of jointly managing several hypotheses is discussed below (several techniques are described), let us now suppose that each hypothesis is managed separately.

9 Technique known as "per-state stack" and described into more detail in Section 10.6.6.

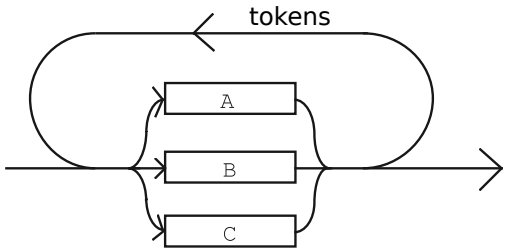


Figure 125: Re-entrant lexicon decoder with token passing (adapted from [Young *et al.* 1989; Figs. 5 and 7a]).

illustrated in Figure 125. Hypotheses inside this lexicon estimator have to carry some trace-back information with them to be able to retrieve the best paths (or to construct a word-graph) afterwards. This information could be implemented in the form of pointers to the nodes of a trellis, but it is better to abstract this concept in the form of a "token" value which is manipulated by means of a predefined API in order to allow a better decoupling between the lexicon, the LM and the trellis managers.

token passing

The token passing approach [Young *et al.* 1989] generally consists in using values representing hypotheses information which can be moved through the lexicon network and which can be combined when different hypotheses are joined in a given state. The idea of using a separate data structure for maintaining back-pointers in a search procedure has probably been reinvented many times and their generalization naturally arises.

Although the term "token passing" seems linked to the concept of re-entrant lexicon, we cannot assure that the usage of the term "token" through the vast literature is limited to this particular scenario and we will rather consider that it can be referred to the actual representations of hypotheses inside a decoder, as opposed to the logical description of these hypotheses, to easy what could be called a separation of concerns. Indeed, and from the lexicon decoder point of view, hypotheses contain three types of information:

- scores which can be modified by the lexicon decoder;
- identifiers to determine if different hypotheses in the same lexicon network state are competing or not; and
- other information which is basically ignored by the lexicon decoder and which is usually devoted to bookkeeping purposes. This information is used to retrieve the best output sequences, to construct a word-graph and so on.

Two different search organizations have been described so far in this section: one the one side, the LM-conditioned search manages a set of lexicon network decoder instances each one attached to a different LM context. On the other side, in the re-entrant lexicon with token-passing a sole instance exists. Indeed, they basically can be seen as implementation choices to perform the same thing with theirs pros and theirs cons.

Let us now see other different scenarios that seem useful to show that other type of information can be required to distinguish among lexicon network hypotheses associated to the same state: Section 11.1 describes how a DAG can be constructed, from a frame sequence, as

the first stage of a two-stage decoder. In this case, a new a lexicon decoder instance is created for each possible starting time position. But, in a complete analogy with the previous case, it is also possible to construct a sole lexicon decoder where each hypothesis is associated a time index or "frame boundary" identifier to distinguish several hypotheses in the same lexicon network state. This "time/input vertex" field" plays exactly the same role as the "LM identifier" from the previous re-entrant lexicon example. Both approaches constitute also a matter of taste regarding implementation. Indeed, the approach consisting in estimating together the set of words starting in a given time index is the other classical way of structuring the search space of time-synchronous one step dynamic programming based decoders [Ortmanns and Ney 2000]. The fundamental difference between the classical time-conditioned decoders and the most equivalent decoder described in this work is the factorization of this decoder into two main parts:

time-conditioned

1. a DAG generator where arcs are labeled with the likelihood of observing a given word in the signal segment represented by the arc. This is the main subject of Chapter 11 where the extension to context dependent units is described and is mentioned here to remark that DAG vertices are not necessarily related with time positions in a one-to-one way;
2. a decoder which can receive as input a word DAG as those described in the previous point.

Indeed, Chapter 8 described several recognition engine configurations based on these modules illustrating the great flexibility of this approach w.r.t. the more classical and seemingly ad hoc descriptions of time-conditioned implementations.

We have seen how two different types of information can be used to distinguish hypotheses associated to the same lexicon network state. But there are many more possibilities depending on the actual use of the lexicon decoder, the following list is not exhaustive:

- the identifier of a vertex from the input DAG. This vertex indicates where the lexicon word hypotheses have started. In this way, the same lexicon decoder instance can simultaneously estimate all segments starting at a given point (a particular case is the time index of a frame sequence in ASR). This type of token is used by graph generators described in Chapter 11;
- a tuple $\langle t, ctxt \rangle$ composed by an identifier t of an input DAG vertex, as in the previous case, and a value $ctxt$ to distinguish different left and right context dependent neighborhoods useful to include across-words context dependencies to the previous case;
- a LM state identifier, which corresponds to the use of lexicon decoders by one pass LM-conditioned search decoders described in Section 12.3;
- a tuple $\langle st, arc \rangle$ composed by a RTN state st and another identifier arc representing a sequence of arc expansions. This case corresponds to the previous LM-conditioned extended to use RTNs in one of the two *classical* modalities explained in this work to

deal with these models. In this case, the "unlimited dynamic expansion approach";

- a tuple $\langle st,v \rangle$ comprising a RTN state st and an input dag vertex v. This case corresponds to the other modality to apply RTNs using what is known as tabular approach.

Note that this information can be partially or completely removed from the lexicon network state hypotheses as far as a different lexicon decoder is used for each possible value from the fields removed from the token: when these values are the same for all states in the same decoder, it is unnecessary to distinguish them. When using such an implicit representation, it is common to perform a mapping from these values to the lexicon decoder copy associated to it. For instance, RWTH decoder based on the LM-conditioned and tree copy approach relies on hash tables for this purpose [Sixtus 2003].

10.2.4 Relationship with across-word context dependency

Context dependent units (see Sections 4.2.1 and 7.6) can be used in a continuous recognition system in two different ways:

WITHIN-WORD dependent units recognition: sub-word units in the middle of words are modeled by context dependent units, but word boundaries ignore this dependency and use context independent units;

ACROSS-WORD context dependency recognizers also take the context dependencies of sub-word units of neighboring words into account. For instance, when using triphones, the right context of the last sub-word of a word may influence the left context of the first sub-word of the next word.

Across-word decoders usually take into account two different types of word transition:

COARTICULATED transition. It is important to remark that coarticulation is not limited to the proper pairing of right and left contexts of consecutive context dependent units. It can also be used as an umbrella term which includes other assimilation[10] phenomena such as the disappearance of one occurrence of certain phonemes which appear simultaneously as the last one of a word and at the beginning of the next one. Another example is the *liaison*[11] where the final consonants of certain words can be pronounced or not depending on the phonetic context of the next word, specially in French. These complex coarticulations can be modeled with the basic approach of establishing a set of contexts, as illustrated in Figure 126;

NON-COARTICULATED transition usually appears when there is a (even short) pause between words, although we have to remark that come coarticulated transitions allows very short silence models between them.

10 See http://en.wikipedia.org/wiki/Assimilation_%28linguistics%29.

11 See http://en.wikipedia.org/wiki/Liaison_%28French%29. This is a particular case of Sandhi, already discussed in Section 2.8.

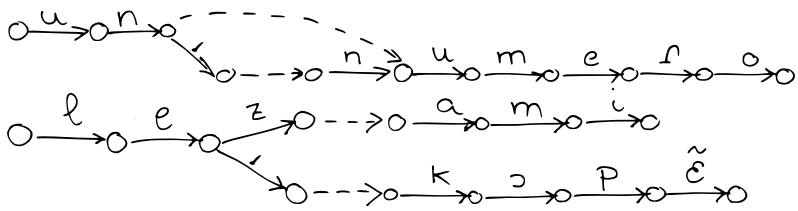


Figure 126: Complex coarticulations: on the top, phonetic transcription of "un número" ("one number" in Spanish) showing the possibility of assimilation. On the bottom, an example of French liaison after french word "les" which can be pronounced either as "le" or "lez" depending on the next word (in this example, "les amis" and "les copains"). Dashed arrows represent connections between lexicon decoder outputs and lexicon decoder entry-points.

This brief section is only intended to discuss how the lexicon estimator API can be modified to deal with across-word modeling, more details on context dependent models and the implementation of across-words modeling is discussed elsewhere.

The transducer composition approach provides a flexible and elegant way to face this feature by creating a transducer modeling phonological rules from context independent to context dependent units [Riley *et al.* 1997]. Dynamic search decoders usually tackle these dependencies by hypothesizing several right-contexts at the end of words (leading to the existence of several fan-out arcs for a given word) and by offering several left-contexts at the beginning of the lexicon network (connected by fan-in arcs) [Sixtus 2003]. In order to allow both coarticulated and non-coarticulated across-word transitions, they also include a fan-out specially devoted for non-coarticulated transitions which includes a silence, as illustrated in Figure 127.

Usually, the sequences of sub-word units associated to different left context of words only differ in the first sub-word unit, the rest is common. It is therefore desirable to use the same decoder and to combine hypotheses incoming from different left contexts, which is known as "Recombination After the First Phoneme Generation" [Sixtus 2003; Section 7.1]. The lexicon decoder interface can be adapted to deal with across-words context dependency in a LM-conditioned one-pass decoder as follows:

- allow the lexicon decoder emit outputs associated to different fan-outs. Each fan out corresponds to an hypothesized right-contexts which is given, for the case of triphones, by the central unit and right context, some of them can be grouped when the left contexts of the corresponding sub-word models are tied;
- enable several entry points associated to left-contexts where the global decoder will introduce the tokens in the re-entrant network approach. Each entry point is a fan-in which is identified by the sub-word central unit and the left context. They can also be grouped due to model tying.

Although the number of different entry points, in the worst case and for the case of triphones, is the square of the number of basic units, it can be greatly reduced by tying similar models.

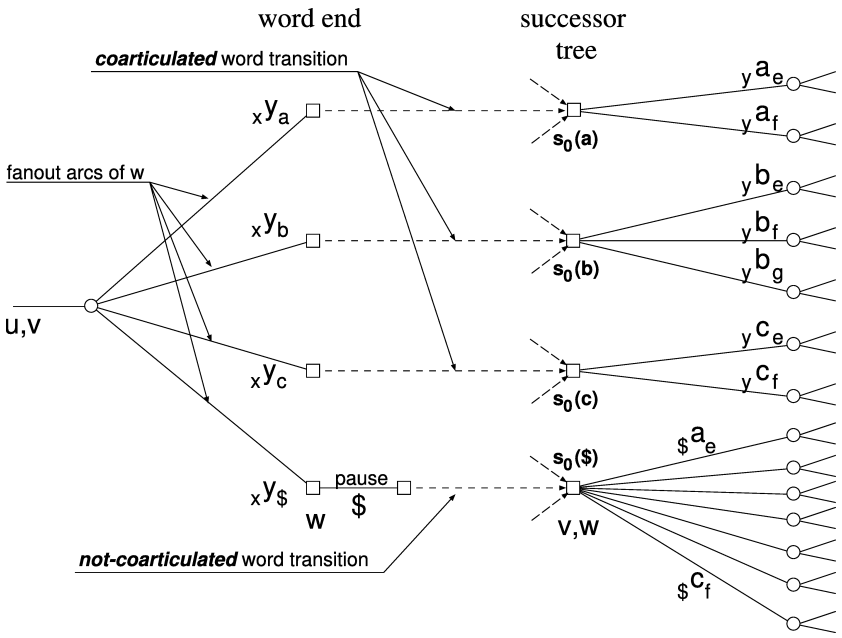


Figure 127: Figure taken from [Sixtus 2003; Fig. 5.2] illustrating the word transitions during across-word search.

It is possible to consider only the left or only the right context. In that case, the number of entry points in the worst case is the order of the number of sub-word units.

A way to deal with context dependency managed by the search procedure itself (instead of being explicitly modeled in the lexicon network) is proposed in [Brugnara 2003]. It is also based on the idea of using several entry and exit points which depend on the context and which are managed transparently by the decoder. Another alternative (but similar) approach consists in placing this context information in the tokens (as proposed in Section 10.2.3) which are managed by the search engine: tokens with a different context value are managed separately (they do not compete among them) but, in this case, the context value is used to select which transition to select: only sub-word models with the left context matching this value. The context value is replaced by the right context of this sub-word model when exiting from it.

The case of a two-stage/time-conditioned decoder is slightly different to the previous LM-conditioned search because the DAG created by the first stage distinguishes context dependencies by creating several vertices associated to the same instant or sequence position. Although it is possible to reuse the same lexicon decoder to produce several right context versions of each word, it is no easy to directly re-combine hypotheses differing in the left-context since they are associated to different vertices. The use of null transitions in the produced DAG may mitigate this problem. The extension of the graph generation algorithm to the case of across-words context dependency is discussed into more detail in Section 11.1.5.

10.2.5 Relationship with pruning

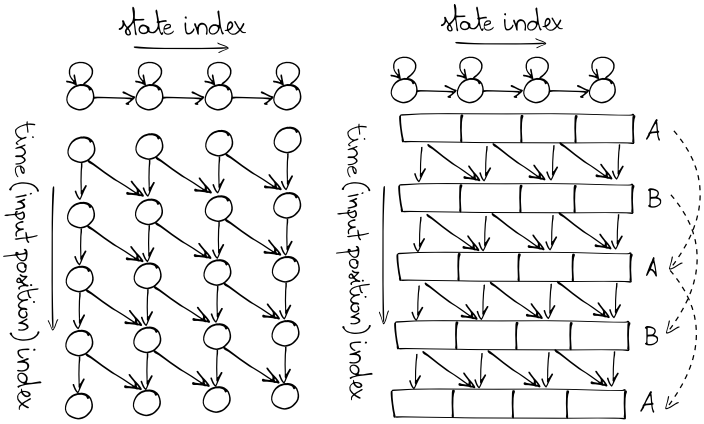
Pruning basically consists in removing some active hypotheses which will not probably lead to any of the best solutions of the problem. This makes sense in decoders which maintain a set of active hypotheses. There exists several pruning techniques. They are described into more detail in Section 10.6.1. Since we are now dealing with the lexicon network interface, let us see how pruning may affect it:

- methods modifying or producing a new estimator (e.g. such as **`edge-step`**) may receive information to perform pruning. Receiving pruning object would allow different lexicon decoders to share this information in order to use more global criteria and flexibility to specify which active states are to be removed. On the other side, this poses problems when used from several threads;
- it is important to determine whether or not a given segment or lexicon estimator is "alive", meaning that it contains some active states. A segment estimator without active states can be removed and other parts of the recognition system may release some resources in response to this notification. For instance, graph generators of Chapter 11 may send a `no_more_out_edges` dataflow message from a vertex whose lexicon decoder becomes dead;
- when a lexicon decoder maintains hypotheses associated to several contexts, it is possible that the global recognizer determines that some one of those contexts are no longer required. A new method to remove hypotheses associated to those contexts would be useful to avoid some unnecessary computation;
- a possible way to implement partial back-tracing is based on determining which tokens are used by some active lexicon decoder state. This techniques is usually activated periodically and tokens no longer referenced serves to determine which part of the output can be emitted. An alternative to this method consists in modifying the interface with the tokens passing management in order to explicitly notify when a given token is no longer referenced inside the decoder, but the expected overhead makes it much less attractive.

10.3 CLASSICAL VITERBI IMPLEMENTATIONS

This section introduces some classical[12] implementations of Viterbi decoders for HMMs and frame sequences. All these implementations correspond to the same dynamic programming algorithm which computes a multi-stage trellis as illustrated in Figure 128a. Every stage corresponds to a boundary between adjacent frames, and each decoding step takes another frame into account. This is a particular case of decoding an input DAG whose vertices represent frame boundaries and whose edges are labeled with frames.

12 The ideas, techniques and tricks described in this section are common of folk wisdom. Although we have tried to properly cite the relevant particularities, the basic techniques are described as in many well known reviews on HMMs.



(a) Multi-stage trellis **(b)** Interleaving two data structures

Figure 128: In **(a)** multi-stage trellis for decoding a frame sequence w.r.t. a finite state model or a HMM, and **(b)** how two data structure instances can be interleaved when decoding (frame) sequences.

Nevertheless, it is important to remark that the implementations described in this section basically ignore the possibility of input DAGs and are specific for sequences using ephemeral or destructive updates. Most of them are based on the use of two alternating representations of the set of active states (as illustrated in , Figure 128a) although the reverse topological traversal may update a sole data structure without requiring two separate copies.

Each Viterbi step consists in computing the set of active states associated to a trellis stage given: 1) the set of active states from the previous stage, and 2) the probability of generating the observed frame, which is computed (some times on demand) for every type of HMM state emission probability. From an implementation point of view, this can be performed in two different ways:

PROPAGATION uses outgoing transitions from the origin states;

AGGREGATION determines, for an hypothesis in the set of active states, which hypotheses are required by using incoming transitions.

This distinction remembers us the difference between push and pull approaches to sequential dataflow architectures (see Chapter 8). Other possible nomenclatures are incoming/outgoing, expanding/consulting or even forward/backwards look-ups.[13]

It is important the idea of maintaining only the HMM hypotheses with a non negligible score, since the number of active hypotheses is usually much lower of the total number of HMM states. In this way, the first option is better when the set of active states can be iterated. The propagation approach is more suitable in this case and most implementations are based on it.

13 We prefer to avoid the last proposal because they can be confused (in spite of finishing with "look-up") with forward and backward algorithms.

Note that most of these approaches can be used "as is" to implement complete recognizers based on the static network approach. Nevertheless, the basic implementations described here assume that the finite state network is not modified during decoding, which is not the case of some on-demand static network approaches. The management of null transitions is not described here to simplify and to better focus the exposition. Null transitions are discussed in Section 10.4.

10.3.1 Array swapping

Without loss of generality, we can assume that state identifiers are integers which can be used to index a linear array. A drawback is that only relatively small models can benefit from this approach.

If we do not apply any type of pruning, there is no much difference between propagation and aggregation approaches: a double loop on states and incoming/outgoing transitions associated to each state can do the work. The aggregation approach is probably better to simplify the maximization and to reduce the number of writes.[14] The representation of models has to be adapted to efficiently traverse incoming or outgoing transitions. Another difference between both approaches is the fact that the forward propagation requires an additional loop to initialize the destination array.

The propagation approach is better suited for the use of an explicit management of active states together with pruning techniques. In this case, the loop for initializing the destination array can be avoided and replaced by the use of time-stamps: a value is stored in each array cell indicating *when*[15] this value has been last written. This technique automatically and implicitly erases all array positions in constant time in the next Viterbi-step. Another problem of using arrays associated to the states of the automaton is that we have to traverse the origin state looking for active states. The obvious solution is to store the active states contiguously[16] which can be efficiently traversed. In this case two different alternatives can be chosen:

- active states are stored in the contiguous list whereas the array of length number of states is used to point to the corresponding position in the former list and serves to detect whether or not this state has already been activated and to locate it;
- the list of active states indexes the array of length the number of states which contains all the information.

The first approach is much better since it can be further improved by realizing that the larger vector is no longer necessary once the set of active states has been the created. In this way, one array instance suffices. Moreover, this approach scales much better in terms of memory usage from sequences to DAGs.

14 And possibly better to parallelize on multi-core architectures due to cache coherence issues if we divide the destination states by cache pages.

15 In which Viterbi-step and, more precisely, in which vertex of the input dag. Most authors prefer to talk about frame index, but frame boundaries and not the frames themselves should index trellis nodes.

16 For instance, in a vector, although we have opted for using a list of vectors like in the internal representation of **`deque stl`** containers. The use of linked lists seems to us too inefficient.

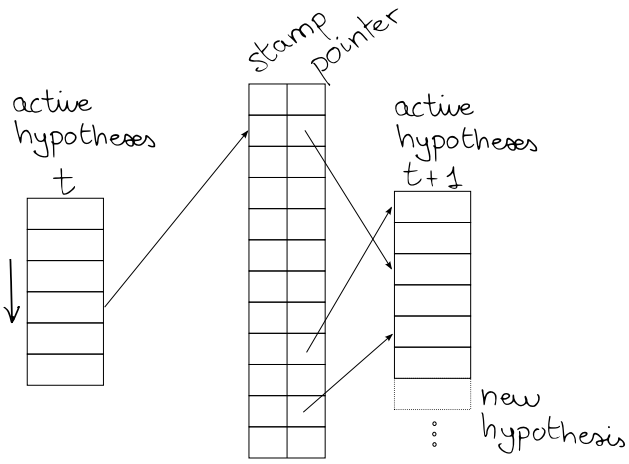


Figure 129: A sole hash table suffices for the hash swapping approach: the set of active hypotheses at time t is traversed sequentially, so we do not need a hash table. Then a new hypothesis is produced, a hash value is generated and the stamp value in the corresponding bucket is used to determine whether this bucket is free or not. If a collision is produced, another index is obtained. In case the key was already inserted, the new hypothesis is combined with the previous one. New hypotheses are pushed back in the active hypotheses vector associated at $t+1$. Active hypotheses vectors contain records composed by the lexicon state id, a score, and a pointer to a trellis node.

10.3.2 Hash swapping

The problem of array swapping, when used together with pruning techniques, is the size of the data structure, since the number of states is usually much higher than the number of active states. This problem is aggravated when several instances of the decoder are simultaneously required. The array is being used as a dictionary or map data structure whose size is the size of the "universe". Other data structures can be used to implement dictionaries with a size proportional to the number of inserted elements. Hash tables seem a good choice, since we can estimate in advance an upper bound on the number of active states and, therefore, choose the proper hash table size to avoid rehashing.

Open addressing collision resolution is well suited for this purpose because we are only interested in a combined search/insertion operation and individual deletions are never required. The same time-stamp technique can be used in this case. Also, the same alternatives as those described in the array swapping case can be used to represent the set of active states which has to be traversed in the following Viterbi step:

- the hash table contains the active states, the main advantage is a subtle increase in data locality when accessing destination states;
- each element of the hash vector contains an index/pointer to the active state representation stored in a growing list of active states, as illustrated in Figure 129. The obvious advantage is that just one hash table is required.

just one hash table

The possibility of ignoring collision resolutions

A well known trick to avoid the cost of resolving collisions during searching in the hash table consists in remembering, at each cell of the hash table, the last inserted hypothesis. When this hypothesis is not the one we were looking for, we simply create a new active state and overwrite the hash cell.

The clever reader may rapidly observe that this trick just "does not work" meaning that some active states can be duplicated when updating a hypothesis that was not the last one. Fortunately, this has no consequences in some decoders *unless* some special token recombination techniques (for instance, to store the N-best hypotheses arriving to each trellis node) were required. It is important to know that this trick cannot be applied in all cases and to measure, when it can be applied, if the cost avoided during the search is worth the additional cost of duplicated hypotheses.

10.3.3 Reverse topological traversal (piggy-backed array)

In previous implementation techniques, two copies of the same data structure were alternated. Fortunately, only the set of active states have to be duplicated and not the entire arrays nor the hash table.

The reverse topological approach, also known as "piggy-backed array" approach,[17] differs from previous approaches in that it only requires a copy of the same data structure which is updated in the Viterbi decoding step. Its use is therefore limited to deal with sequences, as with the other classical approaches discussed in this section.

In order to use the same data structure as the origin and destination of active states, we must assure that active states are never consulted after being updated. This can be easily assured when we work with an acyclic topology, since a traversal in reversed topological order satisfies this property. Similar ideas are used to deal with null transitions and with error correction edition operations (see Section 10.4). An analogy of this reverse traversal is the process sweeping or raking dry leaves from the ground: if you proceed in forward direction you leave traces of your steps. By raking backwards you remove your steps.

An exception to the acyclic property are self-loops, which can be considered a special case that can be easily solved by means of an auxiliary variable.

This technique is very easy to implement when using an array without distinguishing active states. When explicitly managing sets of active states, we need to maintain the list of active states in the same reversed topological order to easy the traversal. This last approach can be easily achieved in some particular topologies as is the case of linear and tree lexicons. These lexicon types are described in Section 10.5. Some algorithms related to the reverse topological traversal, for these particular topologies, are described in Sections 10.8 and 10.9, respectively.

17 We have adopted this nomenclature from [Nguyen 2002]. This idea (which seems so obvious a posteriori) has, no doubt, been reinvented many times: we rediscovered it and, more recently, it seems to be proposed as a novelty in [Lifchitz *et al.* 2006b; Section 2.2].

10.4 DEALING WITH NULL TRANSITIONS AND EDITION OPERATIONS

Null transitions

Null transitions are not associated to observed data, so they have to be performed in the same stage of the multi-stage dynamic programming trellis. It is important to notice that loops consisting of null transitions are usually not allowed, although some decoding techniques could deal with them in some particular cases: when transition costs makes scores worse in such a way that the effect of these loops can be either 1) pre-computed or, alternatively, 2) when the decoding process can be assured to stop after some iterations due to pruning. We will simply suppose that loops of null transitions are not allowed.

loops vs no-loops

The best way to deal with null transitions is in the **`vertex-step`** method that is usually applied after processing all the incoming edges. A naive implementation of this procedure would traverse the outgoing null-transitions of an hypothesis before the same hypothesis is improved again and again. This can be quite inefficient since the set of outgoing null-transitions is traversed several times.

A technique described in [Jelinek 1998] proposes to obtain an ordering of states such that updating outgoing null transitions following this order assures that no state will be updated after being used as origin. Assuming that loops made of null transitions are not allowed, the *longest* distance from each state to another one, following only null transitions, is well defined and can be efficiently computed. This distance is known as the "level" of the state. It suffices to process states from higher level to lower level.

This problem has a little drawback when using a list of active states, as is the case of classical Viterbi decoders, because the order of insertion in this list depends on the destination of transitions from states visited when traversing the former list of active states.

We can do better than naively sorting the list by observing that, in practice, the number of levels is not very high making it feasible to apply bucket-sort [Cormen *et al.* 2009]: It suffices to have a list of active states for each level. In this way, each new state can be placed at the corresponding list in constant time. The use of a hash table with pointers to locate active states makes the management of levels simpler: they are only used when "activating" the state.

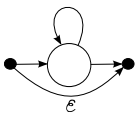
Each list of active states is traversed from higher to lower levels. This traversal, which only applies null transitions, can activate new states as well. Fortunately, the new states will always be situated in strictly lower levels that have not been visited yet. We are not aware of any actual decoder using these ideas and have not seen that described in the previous literature. Indeed, quite recent works such as [Kisun *et al.* 2009] and [Aubert *et al.* 2013; Section 4.1] rely on the recursive traversal, which is the inefficient version described before.

The recursive approach could be better when the percentage of null transitions, their levels, and the respective fan-outs is low. [Kisun *et al.* 2009] also proposes to replace the recursive traversal by the addition of new null transitions associated to null-transition chains. In this way,

only one level of exploration suffices. Other works such as [Phillips and Rogers 1999] propose the use of Dijkstra's shortest path algorithm on the graph of null-transitions using a heap although, in this case, the network structure is computed on demand (which probably prevents the pre-computation of levels of states) and we do not know if there can appear cycles. A similar approach based on Bellman-Ford algorithm is described in the next section.

Relating the use of null-transitions in the dynamic search approach, these transitions are usually placed in some very particular places which makes them very easy to manage. For instance, null transitions are usually employed to model back-off smoothing count based n-grams (see Section 9.2). These null transitions could be tackled as described before, but it is much more efficient to treat them as *failure transitions* which, in turn, makes these models exact implementations of the backing-off technique. Decoders using these models by means of a generic API are not at all aware of the existence of these transitions.

exact implementation of back-off



Another type of null transitions which admit a special treatment is an extension of senones (Section 7.3.1) described in Section 10.8. Let us see now how to deal with error correction edition operation where the case of deletions is essentially equivalent of that of null transitions.

Error correction edition operations

This section reviews how Viterbi decoders can be adapted for edition operations (insertion-deletion-substitution[18] error model) in order to perform error correction decoding. Note that operations can be modeled in the on-demand transducer composition by using a corruption transducer [Mohri 2003]. Another technique to avoid error correction decoding consists in pre-computing an automata which recognizes the input string allowing up to a fixed number of edition operations [Schulz and Mihov 2002].

We will rather consider how to deal with the edition operations in the search procedure. Figure 130a describes a Viterbi-step from one stage of the trellis to the next one. The trellis is an acyclic multi-stage graph. Substitution operations can be handled very easily, in the same way as described in this figure, by traversing transitions and computing the cost of these operations.

The case of insertions is different: In order to interpret that a symbol from the sequence has been inserted, we can stay in the same state (updating the score with the cost of the insertion) instead of using the symbol to perform a transition, as depicted in Figure 130b. Insertions and substitutions are applied at the same time and the hypothesis with the best is kept.[19] Note that in both cases the trellis remains acyclic.

Relating deletions, we have to consider that several consecutive symbols may have been deleted, so that several consecutive transitions may be performed without consuming input symbols but modify the trellis data structure in the same trellis stage k, as shown in Figure 130c. This problem is similar to the case of null transitions from Section 10.4.

18 As stated before, transposition is not taken into account.

19 In general, a token recombination is applied, equivalent to the sum of semiring values. This operation can be used to keep the best hypothesis to create a word graph.

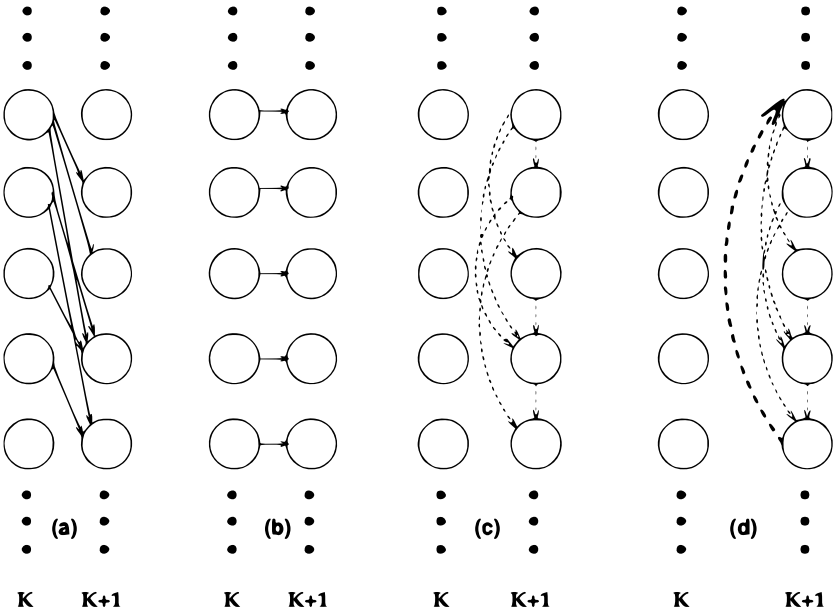


Figure 130: Figure taken from [Amengual and Vidal 1998] showing two consecutive stages from a trellis with: (a) substitution and proper FSM transitions, (b) insertion transitions, (c) deletion transitions in an acyclic FSM, and (d) deletion transitions in a cyclic FSM. Edges are actually labeled with symbols.

It can be solved in the same way when the model is acyclic, since a topological order traversal suffices to assure that no origin vertex will be used later as destination.

The case of a cyclic models transforms the current stage into a cyclic graph, as illustrated in Figure 130d. [Amengual and Vidal 1998] proposes three approaches:

- use Dijkstra's algorithm, an obvious solution to find the shortest path in a cyclic graph without negative weights [Dijkstra 1959; Cormen *et al.* 2009];
- apply Bellman-Ford by repeating the procedure until no score updating is produced. Its expected performance depends on the ordering states are traversed;
- a refinement of the Bellman-Ford approach can be applied by first pre-computing a *nearly* topological order with Depth First Search (DFS) algorithm. DFS can be used to detect back-edges [Cormen *et al.* 2009]. The states are traversed in this order and, whenever a back-edge is used and the destination score is improved, the traversal pointer is changed and continue again from this improved state (which was situated before).

We can easily realize that a sole Viterbi step involving deletions could activate all states reachable from a given state unless we restrict the maximum number of active states with pruning techniques. The three solutions reviewed before can be combined with the pruning methods described in Section 10.6.1. The third approach (based on a refinement of Bellman-Ford) has to reconcile the capability to traverse

only the set of active states (a tiny subset of all states) with the idea of traversing states in a predefined order, but note that this is the same problem that was discussed when trying to apply the idea from [Jelinek 1998] to deal with null transitions by associating a "level" to each state and traversing states from higher to lower level. It suffices to store every new active state in a list corresponding with this value (i.e. to order states by means of bucket-sort [Cormen *et al.* 2009]).

The adaptation of these techniques from parsing sequences to parsing a general DAGs is very easy, as described in Section 10.2.2 and illustrated in Figure 123: it suffices to deal with substitutions and insertions in the **`edge-step`** method. The case of deletions (and possibly also null transitions) is better tackled in the **`vertex-step`**.

An error correcting decoder specialized for tree models, based on the topological ordering, is proposed in Section 10.9.5. This decoder can be used to obtain input word DAGs from input phone (or grapheme, etc.) DAGs, as described in Section 11.2. It can also be adapted to lexicon networks (Section 10.10).

10.5 LEXICON ORGANIZATION

Lexicon is composed by a finite set of lexical items (usually words) where each one has one or several descriptions which are basically finite sequences of sub-words units (a finite alphabet). The lexicon is modeled by means of a weighted finite state model. Since the language is finite, the lexicon model is acyclic. This model is known as "lexicon-network", but two particular cases deserve an especial attention:

LINEAR/FLAT lexicons: are linear sequences of sub-word models (one per transcription). Note that, if the model is expanded, the result may be not linear;

TREE-LEXICONS are obtained from the entire set of transcriptions of the lexicon by sharing common prefixes. The vast majority of decoders are based on tree-lexicons.

Although they are particular cases of the more general lexicon-network, some specific properties of these simpler models are taken into account by some decoders to anticipate the inclusion of LM probabilities, to simplify the manipulation of tokens and so on.

Relating the complexity of a lexicon model, it is important to remark that the reduction in the number of transitions and states in different lexicon organizations has to be measured both in the representation but also in the number of active states.

comparisons can be unfair

The cost of processing each active state may differ depending on the implementation, and some designs take profit of special topologies. A numerical comparison of active states between models using different topologies is clearly *unfair* since an active state in a network lexicon may correspond to several ones in the linear case. Some times, more active states is partially compensated by the fact that the corresponding decoder algorithm can process more hypotheses per second and, perhaps, because their use is simpler (for instance, when managing the past history).

This is the reason why the best way to compare them is to measure the relationship between the resulting WER and the RTF which, as can be observed, depends on other parts of decoder. When these results are not significantly different, we should take other issues into account such as the memory consumption and even the simplicity of the implementation and design. Fortunately, it is possible to compare the speed of different implementations associated to the same lexicon types.

10.5.1 Linear/flat lexicon

In a linear lexicon, each sub-word sequence is modeled with a linear sequence of sub-word units, although it is no longer linear when expanding those models with their respective HMMs.

The term linear lexicon is sometimes extended to cover the case when the different variations of each lexical entry (alternative pronunciations in ASR) are modeled independently. In this last case, we have rather a set of lexicon networks, one per word. Linear lexicons are not the most widespread type of lexicon representation. Their main drawbacks are:

Drawbacks

- the size of the models, taking into account the entire lexicon is usually bigger than other representations such as the tree lexicon or the lexicon network;
- even if the entire model associated to bigrams, using a linear lexicon, is smaller than the tree-lexicon counterpart (without applying minimization algorithms), the decoding cost is higher;
- the LM transitions need to be computed for every word with no (e.g. acoustic in ASR) evidence from the observed signal;
- most part of the computational cost of a lexicon decoder is spent in the initial sub-word units where tree lexicons share most states (90% of the search effort is in the 2 first phonemes) [Ney *et al.* 1992]. In this way, linear lexicons are vastly less efficient.

Advantages

The main advantage of linear lexicons is the possibility to apply the exact language model probability from the beginning. They admit simpler specialized decoders (Section 10.8), although there exist similar alternatives for tree lexicons (Section 10.9). In our opinion, it is only worth considering linear lexicons when:

- the system is strongly guided by the language model and the subset of words in every LM context is very limited and very different from one context to another one;
- we can take advantage of some type of evidence from the signal so that the set of words which are likely to begin at every input position can be restricted. An example of this scenario is the case of multi-pass system;[20]
- it is possible to model entire words in such a way that sub-word constituents cannot be shared easily. This makes tree and lexicon networks to finally resemble a set of linear models. In some cases,

20 Note that most multi-stage approaches construct a word graph and do not perform lexical decoding in subsequent passes but only LM re-ranking, we rather refer to systems which refine the search.

depending on the degree of sharing, it would be better to change to linear models. An hybrid approach useful for a two-stage or time-conditioned search may be based on a tree or a lexicon network to perform the search based on simpler models and to use a word-specific search for those words which have shown some evidence based on the first search. Note that this strategy does not necessarily require a multi-pass approach (but does not require the use of linear lexicons either);[21]

- linear lexicons are straightforwardly easy to parallelize since each word has an independent model. We have to observe that tree and lexicon networks also admit very efficient parallelizable versions, as proposed in this work. Indeed, an approach to parallelize dynamic search decoders consists in splitting the lexicon tree/network into independent parts and an extreme case (e.g. we have one processor per word, just a thought experiment) would lead to a linear lexicon;
- many lexical decoders do not recombine tokens, they just keep the one with the highest score. This is perfectly valid when looking for the best solution, but has an impact when the word-graph of best solutions is also requested since, in LM-conditioned one-pass decoders, there is at most one word graph edge incoming to a given node with a given word. This effect is reduced when using a linear lexicon w.r.t. a tree or a network lexicon. The obvious solution is to recombine tokens properly inside the lexicon decoder.

Examples

Let us first observe that n-gram LMs have the property that each LM state is only reachable from a sole previous word.[22] This property can be taken into account to organize the search space for a LM-conditioned one-pass decoder based on n-grams as illustrated in Figure 131. Each n-gram state would have attached its associated lexicon and both can be jointly considered and activated or deactivated on demand, making it easy to cache LM transition probabilities together with active state look-ups on activated n-grams.

This scheme is often modified by including special models for the particular case when we are backing-off to the unigram or, also, when using the 1-best approximation (see Section 10.6.6). The basic idea of this approximation is to mix hypotheses from different LM contexts in the same lexicon network so that only the best one surviving in each state is kept. This is an *approximation* w.r.t. the standard LM-conditioned search.

Let us observe that, in the standard architecture, each hypothesis associated to a different LM identifier is placed in a different lexicon decoder. The 1-best approximation is more catastrophic in a tree-lexicon

21 A different approximation is proposed in Chapter 11 where, instead of replacing the decomposition of a word into sub-words, a re-ranking of the overall scores is performed using ideas proposed in Section 7.7.

22 Note that this property is not exclusive of n-grams but is not generally applicable to any LM either. Note also that this feature does not apply when backing-off to the lowest level in typical representations of back-off n-grams, as described in Section 9.2.

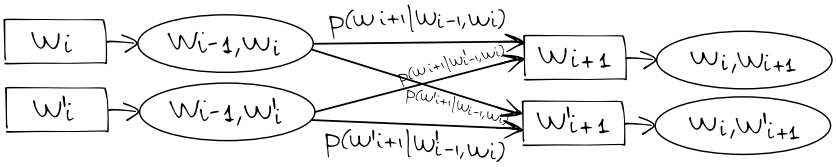
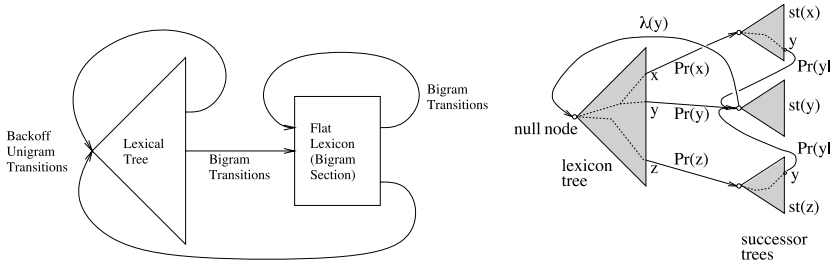


Figure 131: n-gram Four trigrams with their respective transitions are depicted here to illustrate a possible organization with a linear lexicon. A flat lexicon decoder per word is used associated to each trigram. LM transitions are applied before entering each word model.



(a) Flat lexicon combined with tree-lexicon. **(b)** Approach with tree-lexicons.

Figure 132: In **(a)** the combination of a flat lexicon with a tree lexicon is used in Sphinx-II decoder (figure taken from [Ravishankar 1996; Figure 4.4]). In **(b)** a similar approach is modeled using tree lexicons also for bigram transitions (figure from [Antoniol *et al.* 1995; Figure 2]). This last example is notable since an automaton representation of bigrams, with null transition backoffs, is implicit and successor trees already have exact LMs and even LMLA.

network since all hypotheses starting a word compete at the root of the tree. By using a lexicon network this problem is alleviated since only hypotheses associated to the same word are competing and, contrarily to the case of tree-lexicons,[23] competing hypotheses already have the exact contribution of the LM.

We cannot find many recent works describing flat lexicons in the literature. Most decoders are based on tree-lexicons or, less frequently (although more common than linear lexicons) network-lexicons. Some exceptions (including less recent works) are:

- linear lexicons were used in Sphinx-II ASR [Ravishankar 1996; Section 3.2.1] combined with a tree lexicon to incorporate a bigram LM (back-off unigram transitions are estimated with a global tree-lexicon decoder and bigram transitions are based on linear lexicon, as illustrated in Figure 132a). Other works, such as [Antoniol *et al.* 1995], have followed a similar approach for decoding but have used tree-lexicons (Figure 132b);
- more recently, PocketSphinx decoder [Daines 2011; Section 3.4.2] has followed a similar architecture inherited from Sphinx-II. It is based on a three-stage decoding strategy (see Figure 133) where the first stage is based on a tree lexicon approximate search

23 Although there exists the LM look-ahead technique, see Section 10.6.7.

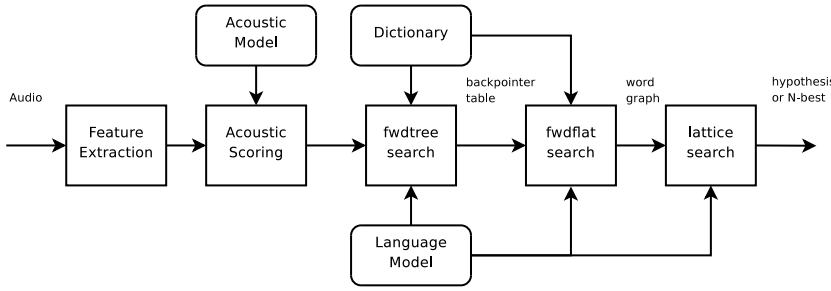


Figure 133: Figure taken from [Daines 2011; Fig. 3.1] illustrating the decoder architecture of PocketSphinx composed by three stages. The first one uses a tree lexicon and is followed by a refinement based on a linear/flat lexicon as in Sphinx-II (see Figure 132) from which PocketSphinx is a successor.

(using 1-best approximation, see Section 10.6.6) and the second staged refines the first one and is based on a linear/flat lexicon;

- a similar approach to Pocketsphinx is described in [Kitaoka *et al.* 2005] where the 1-best approximation is used in the linear lexicon itself in a one-pass approach. A tree lexicon, which is also based on the 1-best approximation, is used in parallel and only a subset of words are recognized by means of the more accurate linear lexicon;
- the VSR recognizer for mobile phones [Varga *et al.* 2002] also employs a linear lexicon according to [Astrov 2007];
- the regularity of linear lexicons can be exploited by highly parallel computer platforms as proposed in [Science and Ravishankar 1993] and, more recently, in [Chong *et al.* 2010] where this part of the decoder is placed in the GPU.

10.5.2 Prefix tree lexicon

Most of large vocabulary Viterbi based recognizers make use of a lexicon tree (or trie) organization. Trees are used to model lexicons at least since [Klovstad and Mondshein 1975]. The advantage over linear lexicons is not only limited to a model size reduction, the number of active states is also reduced. Active states shared at the beginning of words represent more equivalent linear lexicon active states. However, these improvements seem task dependent, [Ney *et al.* 1992] reports a reduction of the static model size of 2.5 w.r.t. the linear lexicon and a reduction factor of 7 in the search effort. The compression factor of 2.5 is reduced to 1.9 when using context dependent triphones.

drawbacks and advantages

Prefix tree lexicons are faster than linear lexicons, but the identity of words is not known until the end (or nearly the end) of the word. The problems of this issue are handled by means of the LM look-ahead technique described in Section 10.6.7. Tree lexicons have several advantages over more general lexicons networks. Most of these features means that decoders for prefix tree lexicons may be simpler:

- in a tree lexicon, the identity of a word is known given the final state, which is not necessarily the case in more general lexicon networks;
- each state, excepting the root, has just one predecessor;
- the above property is specially important because lexicon states with several predecessors require either to combine tokens or to loss word hypotheses just because there exists a more promising word hypotheses, which seems just inadmissible. Prefix tree networks are exempt of this problem;
- tree lexicon networks admit a special ordering of states which makes it possible to store active states in contiguous memory so that they can be efficiently processed without requiring mapping data structures to locate them. Unfortunately, this technique seems mostly ignored despite of the fact that new contributions has been proposed in this regard in this work [España-Boquera *et al.* 2007]. These models and some additional improvements are described in Sections 10.9 and 10.10.

There also exists mixed approaches. For instance:

- tree lexicons used in across-words context dependent units start with a lot of fan-ins making the first state different to a tree, it is more similar to a forest with shared sub-trees. This approach is known as "recombination after the first phoneme generation" (see Section 10.2.4 and [Sixtus 2003; Section 7.1]);
- a mix between prefix tree and linear lexicons is found in [Hauenstein 1993]: the initial part is a tree but the rest of the system is a set of linear/flat models. The rationale behind this design is to share the parts which are computational more costly but to apply exact LM transition scores sooner than in a tree-lexicon.

10.5.3 General lexicon networks (a.k.a. DAWG)

It is obvious that more compact representations than prefix tree lexicons are possible. Although the most general topology would be a general finite state model, the use of a finite lexicon leads to what is usually known as a Directed Acyclic[24] Word Graph (DAWG). It is easy to see the size reduction possibilities w.r.t. tree lexicons from the fact that tries allow the sharing of common prefixes but do not take profit of common suffixes. We can find this observation, for instance, in [Demuynck *et al.* 2000]. Unfortunately, as the authors explain, the gain reduction of their proposal is limited because they also want to guarantee that each valid path in the lexicon network contain *one and only one* word identifier.[25] In this way, the lexicon network is constructed by compacting the suffixes of a tree lexicon by means of heuristics. This remembers us another hybrid tree/network approach even more

24 At least when described in terms of their segment constituent. As remarked in previous cases, this is no longer true when considering the expanded model where these subword models are replaced by their HMM descriptions (in case they are modeled as HMMs and in case they are expanded, neither of these options is mandatory).

25 This seems *too* restrictive as discussed below. A clever reader may observe that even the classical tree lexicons do not fulfill this property!

restrictive since the model is still a tree at the word level and only pronunciation variants are merged in order to be able to determine the word identities just by the final state [Novak *et al.* 2003].

Compact FSA have been used for decades in computational morphology for representing natural language dictionaries [Lucchesi and Kowaltowski 1993]. In particular, perfect hashing automata allows for a more compact representation than the network of [Demuynck *et al.* 2000] since they do not force the paths associated to each word to have a unique labeled transition. Instead, it is possible to attach a value to each transition so that a different identifier is obtained, for each word, by summing the values of the transitions of their corresponding path.

In any case, the size reduction is accompanied by a more complex management of hypotheses during decoding, when using dynamic decoders, since the identity of the state associated to each hypothesis no longer suffices to determine the word that is to be recognized. Token recombination is slightly more complex and an additional field, in the active hypotheses inside the lexicon network decoder, is usually required. The use of DAWGs seems straightforward in the case of static decoders since they usually rely on minimization and other techniques (such as weight pushing), that can be applied normally to these models, to produce an optimized global finite state model.

The trade offs between size reduction and decoding is addressed in [Sixtus 2003; Section 7.6] where the proposal of [Demuynck *et al.* 2000] is analyzed. However, besides the influence of using a lexicon network other than a trie, another difference is present: the system described in [Demuynck *et al.* 2000] makes use of a sole instance of the lexicon network for all hypotheses instead of making a copy for each different LM context as in [Sixtus 2003].

It is also interesting to remark that the advantage of more general lexicon networks is diminished when using context dependent units. On the other side, the capability of DAWGs for joining word suffixes helps to factor the number of contexts at the end off the model when using across-word context dependent units.

theoretical challenges

Lexicon networks have constituted and still constitutes a subject of research to formal language theoreticians. Although minimal DAWGs are compact representations and there are efficient algorithms to construct them, some authors have addressed the problem of reduce the number of arcs instead of reducing the number of states. Moreover, the use of non deterministic model can lead to more compact models [Maire *et al.* 2003] although it is known that the obtaining the minimal non deterministic model is an NP hard problem [Berstel *et al.* 2010]. Other possibilities for allowing a further size reduction include the possibility of including extra words [Câmpeanu *et al.* 2001]. This would not pose a problem to the overall decoder provided these hypotheses do not interfere with the correct ones and provided they can be removed or ignored.

parallelism

A problem type that we have not seen elsewhere in the literature is how to split a lexicon network in a predetermined number of models that could be used in a parallel decoding system as those described in Section 12.1.1. Note that it is simpler with tree lexicons. Observe also that this is not the same as first generating the partition and constructing a network for each partition since the first stage determines

the quality of the second, so that this greedy procedure would be a sub-optimal approximation.

Therefore, lexical tree organization seems a very good trade-off between compact space representation and adequacy for decoding. Their decoders are quite easy to design and probably the gain of using network lexicons is not worth the extra implementation effort. Also, there exists many ad hoc techniques for tree lexicons not described in network lexicons, as is the case of LM look-ahead.[26] These reasons, among others, explain the popularity of tree lexicons. Indeed, there exists few works describing the use of more general lexicon networks. For instance, only two works are cited in [Aubert 2002] when referring to this subject: [Hanazawa *et al.* 1997; Demuynck *et al.* 2000]. Relating the use of non-deterministic DAWGs, a modest number of references can be found, sush as [Georgila *et al.* 2000; Lifchitz *et al.* 2006a].

10.6 TECHNIQUES TO REDUCE THE COST

There exists lots of techniques to reduce the cost. Most of them should be applied over the (overall) global recognizer instead of being limited to the lexicon decoder level. Some notes on the global optimization are delayed to Section 12.1.1. This section describes some of them; others have been discussed elsewhere (e.g. estimation of HMM emission probabilities) and others have simply been left aside.

10.6.1 Active states pruning

Pruning makes sense when hypotheses with negligible probabilities are not actively processed. We have seen that many Viterbi implementations have a computational cost which basically depends on the number of transitions outgoing active states. Pruning techniques only makes sense in these scenarios since they are designed to limit the number of active states.

It is clear that locally optimal solutions do not necessarily lead to the best global solutions because they have only taken into account a part (a prefix) of the input signal. These hypotheses can loose their advantage in subsequent steps while processing the rest of the signal. There exists many different pruning variants but they are generally based on the assumption that globally optimal solutions come from reasonably good local solutions, meaning that the capability of changing their relative advantage is limited in practice. Pruning techniques are mandatory for practical reasons (excepting, maybe, some toy tasks) and, as explained before, they typically follow a law of diminishing returns: a very aggressive pruning has a severe impact in the word error rate; as the pruning becomes more permissive, the error decreases as the decoding time becomes more and more costly to the extent that huge increases in decoding time are required to obtain tiniest improvements in error rate.

26 We do not mean that LMLA is not possible for general networks, they are probably not reported because it is more cumbersome or because network lexicons are not so widespread.

A proper setting of pruning allows for a reasonable decoding speed scarcely degrading accuracy. Nevertheless, a technique which does not require tuning is described below.

There exists two main different criteria to select a subset of active states from those who satisfy a certain condition:

BEAM SEARCH maintain active hypotheses relatively good compared with the current best hypotheses, no matter their number.

HISTOGRAM PRUNING establish a maximum number of active states.

These pruning techniques can be applied at different levels:

LM LEVEL is when the pruning technique is applied at hypotheses after the LM probability is applied, just before entering the lexicon decoder.

STATE LEVEL refers to the pruning applied to active hypotheses inside lexicon decoders. The term "state" is not very clear, but the most usual denomination "acoustic pruning" seems too ad hoc (the majority of work on decoding has been first or exclusively described for ASR tasks).

Note that it is possible to specify other types of pruning by characterizing hypotheses into groups sharing some property which, in the case of dynamic search decoders, can be related to the lexicon search, the LM updating and so on. Some examples can be found in [Pylkkönen 2005], in [Jelinek *et al.* 2001] or in [Huijbregts 2008; Section 6.2.1]. The expansion of new lexicon transitions can be controlled by other type of pruning as is the case of the fast-match look-ahead technique (Section 10.6.2) or the use of "unit deactivation" pruning [Renals *et al.* 1995] which avoids the expansion of some lexicon transitions when the corresponding frame posteriors are quite low.

A way to modify pruning techniques consists in applying a punishment or a reward to the scores when they are considered for pruning purposes, without actually altering what is computed. For instance, in [Kato *et al.* 2010] the number of potential words reachable from a given state is used as an heuristic to help those hypotheses to survive. This technique play a similar role to LM look-ahead (Section 10.6.7).

Some pruning checks may be redundant: when an approximated optimistic estimation of the current score can be computed without effort, it can be computed to check if the current hypothesis can be pruned before actually computing the correct value. This is a waste of time for non-pruned hypotheses, but can be compensated when the a-priori probability of being pruned is high enough. Another pruning possibility is to use cheaper models for less promising hypotheses. This seems less drastic since a pruned hypothesis cannot be recovered anymore, but another one using a cheaper model could gain ground later. The techniques proposed here can be used to detect which hypotheses are less promising, they can be easily adapted to classify them into those which are kept, those which are *degraded* to cheaper models and those who are simply removed. They can also be used to (partially) sort them in order to first use the most promising ones, the rationale behind this heuristic is explained below.

Beam Search

Beam search[27] only expands hypotheses whose scores are relatively close to the best one (see Figure 134). The beam width threshold value is fixed and scores below the best one *minus* this threshold are removed. We have assumed that decoders work with (maybe negated) log-scaled likelihoods so that this criterion is relative in terms of probabilities. Although it is possible to apply this pruning technique once the set of active states has already been computed, it is usually better to apply pruning techniques as soon as possible, i.e. while the set of active states *is being generated*. Since we do not know the best score of the current set until it is too late, we can use instead the best score found for the moment or, if possible, an estimate. In both cases, this value may change in the course of the process.

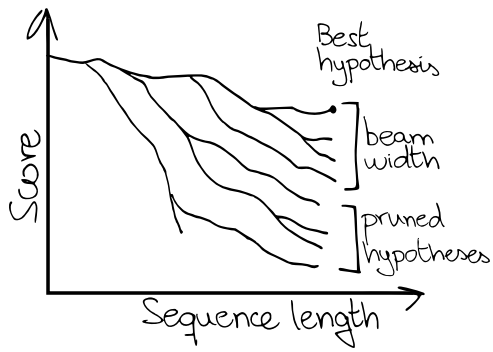


Figure 134: Beam search.

The same hypothesis may survive or not depending on the current value of the best score. Since the best score applied at the moment of pruning a given hypothesis may be improved afterwards, it is possible that some non-pruned hypotheses are no longer valid w.r.t. the final beam pruning criterion.

This is the reason why, once the set of active states of the destination trellis column has been completed, the final threshold value used to prune hypotheses is stored and used to prune *outgoing* hypotheses in the next Viterbi iteration. This also explains why it is important to process first the states achieving the highest scores: they would probably lead to the best current score and, hence, reduce the number of unpromising active states.

A common procedure consists in computing an estimate of the best score either by propagating first the best state[28] or by combining the best score of the previous iteration with an upper bound of the cost of transitions.[29] This approach is known as "Killer heuristic" [Beyerlein *et al.* 1997] or "anticipated pruning" [Sixtus 2003; Section 9.1] [Rybach 2014; Section 6.3.1].[30]

27 According to `http://en.wikipedia.org/wiki/Beam_search`, this term was coined by Raj Reddy, at Carnegie Mellon University, in 1976. We have found a description of this technique in the PhD word of Bruce T. Lowerre in the same year [Lowerre 1976].

28 Or the best ones. This may increase the decoder complexity or limit its design.

29 By taking into account the best LM transition, in the case of LM pruning, or the best HMM transition at sub-word level together with an estimate of the best emission probabilities (computed from the training corpus and kept constant [Sixtus 2003; Section 9.1], or computed from the current frame using cheaper models as in the fast-match look-ahead technique described in Section 10.6.2).

30 In the case of [Rybach 2014; Section 6.3.1], the pruning is applied before computing the acoustic model scores and the hypothesis is compared with a current threshold which does not contain the acoustic score either in order to be more properly compared.

Avoiding setting the beam width

As described before, the beam width must be configured and this may depend on the task at hand. An aggressive pruning means a faster decoding, but has a severe impact in the WER; as the pruning becomes more permissive, the error decreases as the decoding time becomes more and more costly to the extent that huge increases in decoding time are required to obtain tiniest improvements. A very interesting way to avoid this setting is proposed in [Nolden *et al.* 2013]. This technique is based on the relationship between the correspondence of the best solution obtained in a forward and in a backward search using the same models[31] and the presence of misrecognitions. This fact is used by the system to perform a forward and a backward search starting with an aggressive beam in order to iteratively increase the width until both results coincide.[32] Obviously, some computations are reused (e.g. acoustic scores and incremental decoding).

Histogram Pruning

A problem with beam search is that peaks on the number of active hypotheses, much higher than the average, are observed during decoding. Indeed, these picks have been related to some features such as the presence of ambiguities, the beginning of words, etc.

Histogram pruning, the limitation of the maximum number of active states, **`maxHyps`**, is usually applied together with beam search to establish a worst case processing time for decoding. In this way, we can put a higher beam width without risk of keeping too much hypotheses. Note that there are different ways to limit the maximum number of active hypotheses **`maxHyp`** besides using histograms[33] and that this technique can be used without beam search, although some implementations require it. As with beam search, it is possible to limit the number of states a posteriori once the set of active states has already be computed by selecting the best ones using the selection algorithm in linear time [Cormen *et al.* 2009]. It is usually better to apply this pruning techniques while the set of active states is being generated. Let us describe how to easily achieve that. Unfortunately, even recent[34] papers such as [Aubert *et al.* 2013; Fig. 7] describe beam and histogram pruning techniques which only apply these techniques to discard hypotheses when processing the next frame.

Let us describe how to prune by using an actual histogram before briefly discussing other techniques such as the use of heaps. When using an histogram, the technique is used jointly with beam search. A beam width value and the number of histogram bars is required as

31 But properly adapting the lexicon and the other models.

32 As a future work, we would like to study the possibility of implementing this technique and to combine it with the use of an *external* technique (meaning external to the decoder, in the same way as there exists external oversegmenters) to detect which parts of the input signal require a more permissive pruning instead of using the same threshold in the entire signal. A possible way would be to measure the entropy of the posterior distribution in hybrid HMMs based on emissions obtained from these posterior estimations (e.g. HMM/ANN).

33 Therefore, the name of this technique is a synecdoche, since a particular feature of a possible technique to implement this technique has been used to denote it.

34 The publication date of this work is posterior to the time at writing these words.

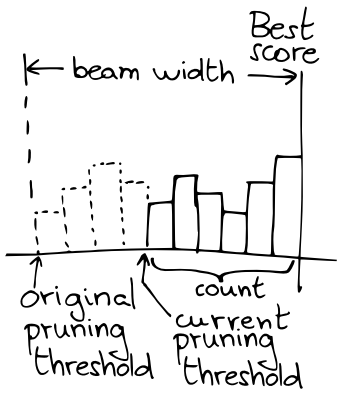


Figure 135: Illustration of histogram pruning once some destination states have been activated. The original pruning threshold was the same as in beam-search. Some histogram bars have already been discarded when the accumulated count of active bars exceeds **maxHyps**. In this case, bars on the left are removed, reducing the count (and hardening the pruning threshold), until it is again lower than **maxHyps**.

configuration settings. An estimate of the best score, in a similar way as beam search, is also required. Every histogram bar stores the number of active hypotheses within a range of scores obtained by dividing the interval between "the best score estimate" and "the beam search pruning threshold" into the number of bars. When a new hypothesis is generated, it is discarded if its score is below the threshold. When it is not discarded, the corresponding bar is increased as well as a count equivalent to the sum of current bars. If this count reaches the value **maxHyps**, the bars associated to the worse scores are removed from the histogram in order to maintain the count below **maxHyps** again. The threshold is always updated to the lowest active bar, as illustrated in Figure 135. This threshold will be used, as in the case of beam search, to discard unpromising hypotheses which survived the pruning. Nevertheless, observe that the pruning can be applied at the current frame as soon as the number of active states reaches **maxHyps** [Llorens and Casacuberta 1999].[35] Two remarks have to be done on this respect:

good recipe

1. the number of surviving hypotheses is usually slightly higher than **maxHyps** since thresholds are applied using entire bars. The more bars the more precision, but the spatial and temporal cost is also increased; and
2. when state are first activated, the corresponding bar is increased. Unfortunately, when the destination state already had a value, we should also decrease the bar associated to the previous hypothesis. This last procedure is usually skipped or simply not detailed in most descriptions of the algorithm. The problem of skipping this part is that the number of hypotheses is overestimated and causes some problems in many cases.[36]

As described above, the last computed threshold is used to discard unpromising hypotheses when traversing the set of active states the in the next Viterbi step. A curious variant is found in RWTH decoder [Ortmanns 1998; Section 3.2] [Sixtus 2003; Appendix C] where the set of active states is generated in two stages as illustrated in Figure 136: in the first stage, active states are created and placed in the same vector

35 Details of this technique have been given by David Llorens in a personal communication.
36 Obviously, we have implemented the correct version in our toolkit!

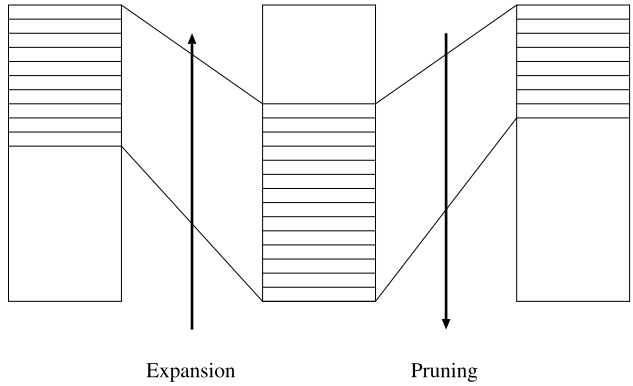


Figure 136: Figure taken from [Sixtus 2003; Fig. 7.9] illustrating how active search hypotheses are generated into two stages: in a first step, new hypotheses are created by traversing the set of active hypotheses from bottom to top. In a second step, the set of hypotheses is traversed from top to bottom and packed again at the beginning of the vector. This last step is used for pruning.

as the original set of active states, but from bottom to top.[37] In a second step, states are copied from top to bottom and the pruning threshold is then used to remove unpromising hypotheses.

novel proposal

The following idea, novel to the best of our knowledge, can be incorporated to histogram pruning *nearly for free*: it consists in assigning a list of active states to some histogram bar groups. The first time a state becomes active, we look for its bar first and locate the state in the corresponding list. Later, when these active states are used in the next time frame, the state lists are processed from higher to lower score. Observe that hypotheses are not in a completely sorted way but are sorted in ranges. This seems enough to obtain a good pruning behavior. This technique can be adapted to the use of several decoder instances[38] (for instance, in the lexicon-tree copies approach) by processing first the lists associated to better score ranges.[39] As the reader can observe, the rationale of all these techniques is to explore first the best hypotheses to make histogram pruning work with hard thresholds as soon as possible to reduce the number of unpromising activated states.

heaps instead of histograms

Another method to limit the number of active states, different from the use of histograms and which do not need to be combined with beam search, is the use of heaps. Assuming that we are maximizing log-likelihoods,[40] we can use a min-heap of size **`maxHyps`** in order to locate in constant time the worst active state whose score can be used directly to prune new hypothesis. A new hypothesis beating

37 They do not overwrite the original set because it is also traversed from bottom to top. Nevertheless, we prefer to avoid unnecessary data structure traversals.

38 Observe that we are processing several decoders in a serial way. This is different from parallelization (execute each decoder in a different thread or process, as discussed in Section 12.1.1). Parallelization makes the sharing of a pruning criterion more difficult.

39 It is convenient, when using several lexicon copies, to use a global pruning criterion. However, since changing the decoder instance too often is bad for cache purposes we can, alternatively, count the number of hypotheses in the best positions of each model and proceed sequentially one decoder at a time.

40 Many systems minimize negated log-likelihoods, instead, to avoid negative values.

the worst one replaces it and the heap property can be restored in $O(\log(\texttt{maxHyps}))$ steps. The heap property does not need to be maintained until the vector is full and, when it is first filled, the heap structure is obtained in linear cost. As an example of the use of heaps, the iAtros recognizer [Luján-Mares *et al.* 2008] uses a heap to store the states from a recognition stage and a hash table to locate them. Both data structures contain pointers to hypotheses. Relating our implementation, we have tried both histogram pruning and heaps, although heaps has only been used to store scores and to provide the a pruning threshold. There exists some techniques based on modifying the pruning threshold in order to *approximate* histogram pruning without explicitly storing information on the best **maxHyps** scores [Aubert *et al.* 2013].

Pruning objects and pruning interface

The idea of pruning objects arises naturally when we want to limit the number of active states associated to a set of decoders: the set of parameters and histogram pruning bars have to be shared by different decoders. But, why to hard-wire this particular way of pruning? Instead of coupling it with decoder implementation, we believe it is worth investigating the possibility of designing a more generic API so that any pruning object can be used interchangeably. Even if we have to control only the number of active states associated to just one decoder, we would like to factorize this behavior and the relevant information into a class in order to increase the modularity. This is desirable for several reasons:

- several decoders may use the same pruning scheme,
- it would be desirable to be able to choose among different pruning techniques for a given decoder, can that be done by changing the pruning object implementation?
- it is possible to use a pruning criterion which involves jointly several decoders.

The proposed pruning interface contains the following methods:

RESET receives a new best score, in the case of using an histogram, it resets the bars and the count,

PROCESS_SCORE score value and update the histogram bars assuming that this state is to be created,

CHECK_SCORE also receives a score value but do not update the histogram bars,

TRANSFER_SCORE receives two scores assuming that one of them replaces the other (already introduced in the histogram).

Another issue related to the use of pruning objects is related to the parallel computation of decoding. Pruning techniques which need to update an internal state (for instance, histogram pruning) cannot be shared across several threads without solving the problem of synchronization and mutual exclusion. Details on how this issue has been solved for our parallel implementation of decoding has been addressed in Section 12.1.1.

10.6.2 Fast-match look-ahead

Fast match look-ahead consists in estimating the next frames to be processed by means of computationally cheaper models (which are usually called "fast match".[41]) in order to help to prune less promising hypotheses [Aubert 1989; Ortmanns *et al.* 1996; Sánchez *et al.* 1999]. The following methods can be used to get cheaper models:

- to reduce the number of models. For instance, using broad phonetic classes in ASR [Aubert 1989], or, more commonly used, using context independent models when the system usually employs context dependent ones (i.e. triphones);
- the use of simpler models to estimate the emission probabilities of HMMs. For instance, use semi-continuous models or reduce the number of Gaussians in GMMs;
- to decrease the number of frames by down-sampling the frame sequence in some way, as proposed in Section 10.6.3.

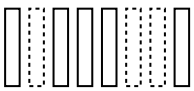
It is possible to estimate more than one sub-word unit ahead. In [Kenny *et al.* 1991] two phones ahead are estimated.

[Nolden *et al.* 2011b] constitutes a notable step in the fast-match look-ahead. In this work, a lower bound on the effect of this technique is obtained by performing a "perfect" look-ahead by using the same models as in recognition. Using this bound, authors conclude that there is not much room for improvement since quite fast techniques already achieve a substantial part of this achievable improvement.

10.6.3 Variable frame rate and frame dropping/skipping

The computational cost of decoders is usually expressed in big O notation[42] where the size of the problem is the sequence length.[43] In some problems, the sequence is a continuous (sampled) signal, as is the case of ASR and HTR. The length could be measured in seconds, millimeters, etc. but this signal is usually converted into a frame sequence by applying a windowing. The number of frames depends on the window advance.

Window advance cannot be augmented arbitrarily, but values used by different systems may differ. In the case of ASR, a window advance of 10ms is very common, although some authors argue that this value is not adequate to characterize rapidly changing parts. Window advance values as low as 2.5ms or as high as 15ms has been reported in the ASR literature ([Zhu and Alwan 2000] and [Astrov 2007], respectively). These values do not affect the asymptotic cost, although they affect the constants involved. That is why it is important to reduce the number of frames, provided this reduction does not affect the quality of the system. The use of a variable frame rate and/or frame dropping has several purposes, the list is probably not exhaustive:



41 Conversely, some authors describe the original models as "detailed match".

42 See `https://en.wikipedia.org/wiki/Big_O_notation`.

43 The case of DAGs is slightly more complex since the number of vertices and edges do not characterize in the same way all problem instances. For example, the number of edges of the transitive closure is relevant in some cases but not in others.

- reduce the number of frames and, hence, the computational load;
- it is also possible to adapt the feature extraction to particularities of the signal. For instance, to provide more detail in regions with a higher number of changes or to provide some way of rate normalization (speech rate in ASR, width normalization in HTR);
- although there may exist better measures to address the problems caused by the noise parts of the signal, the corresponding frames can be simply removed when they are not to be properly tackled by the recognition system. In this case, a hard decision about the presence of noise has to be taken by the preprocessing stages instead of relying on later steps of the system where more information is available, the advantage is the simplicity;
- to diminish the space required to transmit the frames in distributed applications as is the case of client-server based ASR systems (e.g. on mobile devices) where the client receives the speech signal and sends the parametrized frames to the rest of the system placed in a remote server.

The term "frame dropping" or skipping is used in the literature:

- in some cases, to the removal of non-speech segments detected by means of a voice activity detector (VAD). This is usually done in most ASR systems at the beginning and end of sentences even if most of them never use this term to refer to this technique;
- to speed-up the computation of some frame emission probabilities when this frame is similar enough to the previous one [Finke *et al.* 1999; Section 3.6]. In this case, emissions from the previous frame are reused, but the search procedure does not take profit of this similarity to save some computations. Otherwise stated, frames are not removed from the search process;
- some frames are removed when they are classified as less reliable [Duvinage and Parfait 2010] or when they are sufficiently different from the last non dropped frame as in [Zhu and Alwan 2000]. In the first case, the aim is to make the system more robust to noise, the second example can lead to significant computational reductions without a significant degradation of the recognition rate. Another interesting frame dropping approach related to decoding is proposed in [Siu and Chan 2006] where the search process itself is adapted to limit the number of removed frames.

These techniques seem quite task dependent. Unfortunately, we have not found in the literature any work describing how the segment models (particularly, HMMs) can be adapted to variable frame rate or to skipping frames. We can only guess that models are just trained with the same type of preprocessed data to reduce the mismatch between training and testing conditions and that conventional HMM search techniques are applied *as is*. This cast doubt on the role of HMM topologies to model segment durations. An exception where the search process is modified is [Siu and Chan 2006] where the number of dropped frames is controlled by including another parameter in the recurrent equations, which leads to another dimension (as expected and regularly done) in the dynamic programming approach.

We believe that the use of a lower frame rate should came accompanied with a set of models specifically trained for these conditions. The use of models designed for a lower frame rate is complementary with other techniques to reduce the cost of some parts in order to speed-up the search process. For instance:

- a lower frame rate version of the signal can be used in some techniques associated to fast-match models where less precise models are just used to guide the search;
- the same idea can be used in multi-pass systems where: 1) a first pass basically serves to generate a subset of solutions in order to detect which parts of the signal are more reliable, and 2) a more guided search, using more detailed models, is performed in the second pass;
- some systems use cheaper models for less promising hypotheses as is the case of [Huang *et al.* 1995; Section 2.2] where some hypotheses are not completely pruned but are just associated with a cheaper to compute model using an heuristic strategy known as "Rich Get Richer" (RGR).

A related approach is the use of an external over-segmenter which avoids the start or the ending of words in some frame boundaries. These techniques are discussed in Sections 10.6.4 and 12.4. The frame dropping criteria described before can be used to cluster frames, rather than skipping them, and would consequently work as an external over-segmentation technique. Indeed, a new technique to speed up some decoders under the assumptions of external over-segmentation (but also useful for time-conditioned recognizers) is proposed in Section 10.6.9.

10.6.4 External over-segmentation

The difference between segmentation and over-segmentation, as well as the difference between internal and external (over)-segmentation modalities has been explained in Section 3.1.4. Now, the relationship with lexicon decoders is briefly discussed while the use of over-segmentation techniques in general decoders is delayed to Section 12.4 and, for the particular case of input DAG generation, to Section 11.1.4. External over-segmentation for lexicon decoders come in two flavors:

AT THE SUB-WORD LEVEL can be used by lexicon decoders to avoid the computation of transitions between sub-word models. The use of over-segmentation at the sub-word level is described into more detail in the respective sections devoted to specialized lexicon decoders. An example of this kind of techniques can be found in the literature: in [Laface *et al.* 1995; Section 3] the tree lexicon arcs are only activated when the corresponding phonetic boundary is likely to be located at this point. A novel different type of segmentation information, known as "sure frontier" and described in Section 3.1.4, can be used to prune all hypotheses excepting some sub-word transitions. Although the use of this type of pruning is risky, it can be done sporadically in some tasks;

AT THE WORD LEVEL in principle, does not affect the lexicon decoder since it basically affect the global decoder which controls the flow of tokens in the re-entrant lexicon. The information provided by an over-segmenter can be used in several ways: when we know that no word can end after the next frame, the global decoder can ignore the tokens produced by the lexicon decoder (avoiding the use of LM-transition probabilities) or, even better, it can inform the lexicon decoder that these hypotheses are not to be emitted and, also, that new tokens are not expected at the beginning of the lexicon decoder when processing the next frame. Although the lexicon decoder can have a method to perform a Viterbi step and another to generate the output words, it is better to have a combined method, kind of specialization, in order to avoid an additional traversal of the set of active states. Another improvement can be obtained when it is possible to know in advance that several frames are to be processed with no tokens incoming to the lexical decoder. In this case, a novel proposed technique, described in Section 10.6.9, can be used to process several frames simultaneously.

A related technique is the detection of positions and labels classified with a high confidence or reliability. These zones known as "anchor points" and might help to constrain the search (related to "island parsing"). However, we have not been able to find relevant works on decoding exploring these ideas.[44]

10.6.5 Information from previous passes

There exists several recognition techniques which rely on the use of more than one decoding step. The typical multi-pass systems are based on the use of a first pass usually performed with computationally cheaper models in order to narrow the search space to be further explored by more complex and expensive models. The second pass is often performed on a word graph produced by the first pass. Most of these techniques are more related to the overall decoder than to the lexicon decoder and, not surprisingly, it is more appropriate to describe them elsewhere.

But there also exists recognition techniques where subsequent recognition steps involve the use of lexicon decoders or where several passes are more intrinsically related than in typical multi-pass systems.

One of such approaches is the use of a forward-backward search. In [Austin *et al.* 1991], a forward search is performed using simpler and faster models. A backward pass can benefit not only from the information gathered in the previous pass to easily determine which models are to be activated but also to obtain a good estimate of the overall score associated to the partial hypotheses. This kind of information is very useful in $A^\star$ or stack-decoding approaches to design the admissible function.

44 Using these terms in search engines has not led to relevant results. The term "anchor point" is used in [Bourlard and Dupont 1997] to refer to the points where the information computed from different sub-bands is combined. Some words refer very vaguely to anchor points in the area of word-spotting.

There are multi-pass systems where the subsequent passes are not limited to manipulate a word-graph. For instance, the approach from [Nolden *et al.* 2013] mentioned in Section 10.6.1 perform an incremental decoding where some parts are reevaluated starting from the word previous to the position where an error may have occurred.

We believe that techniques different from reusing a word-graph can be envisaged which takes profit of previous Viterbi steps in order to obtain gains beyond the naive reuse of frame emission likelihoods.

It is possible, in time-start decoders as well as in the highly related DAG generation algorithms (the subject of Chapter 11), to apply decoding ideas based on applying a first pruning threshold and a given tree lexicon and to reuse part of these computations when re-evaluating the same frames by means of a more permissive pruning criterion and/or a greater lexicon network:

- the basic idea in the case of increasing the pruning threshold is to not completely forget the pruned hypotheses but to store them although the were not further expanded in the first pass;
- relating the use of a greater lexicon network, the key idea is to consider the reduced lexicon network as a subset of the greater one and using the same state identifiers.
  Hypotheses associated to states of the reduced network which have transitions to the more detailed one are stored attached to the corresponding time frame and a re-evaluation of the system would only take into account these novel transitions.

10.6.6 Bundle-search and word-pair approximation

In one of the common designs for LM-conditioned one-pass decoders, the tree-copy approach, a different lexicon network instance is associated to each LM context. These copies are constructed and destructed on demand when a given LM context has a first active hypothesis and where its set of active hypotheses is emptied, respectively. The problem with this approach is the high number of lexicon decoder copies that may be required, specially for long-span LMs.

This section describes some exact and sub-optimal techniques to reduce its cost. They have in common that they manipulate explicit or implicitly several hypotheses associated to the same state of the lexicon network. The idea of manipulating several hypotheses associated to the same network state was already introduced in Section 10.2.3. We believe that the following techniques can be organized in two different but not completely orthogonal axes:

- no bundle/explicit bundle/implicit bundle: meaning a different decoder per context, a decoder with an explicit list of hypotheses for each context and tokens implicitly representing several hypotheses, respectively;
- exact/approximated bundles.

We have coined these terms (disregarding our remark about Figure 5) because we have not found this grouping of concepts nor a standard nomenclature in any previous review, in spite of the fact that all these techniques already exists.

Explicit bundle (per-state stack organization)

This technique associates multiple independent hypotheses in the same lexicon network state. It is coined[45] as "per-state stack organization" in [Aubert 2002; Section 6.1.2] and is described, at least, in [Alleva *et al.* 1996] [Finke *et al.* 1999], [Soltau *et al.* 2001] [Demuynck *et al.* 2000] [Cardenal-Lopez *et al.* 2002] and, more recently, in [Huijbregts 2008; Chapter 2]. The idea is very well expressed in [Huijbregts 2008; Chapter 2]:

> *Tokens that do not share the same history will not compete for a place in a state, but can occupy a state simultaneously. This means that instead of a token administration, a token-list administration is needed in each state.*

The different hypotheses associated to each lexicon tree node can be represented using different data structures. The choice is critical to perform pruning, on the one hand, and token combination on the other. [Demuynck *et al.* 2000; Section 5.3] describes two alternative implementations based on hash tables and lists sorted by the LM identifier. In both cases, the efficient location of tokens that are to be combined is privileged. Lists sorted by the score are described in [Huijbregts 2008; Section 6.2.1], where are more appropriate to perform pruning. Another worth mentioning alternative is the use of heaps, as described in [Alleva *et al.* 1996] [Finke *et al.* 1999]. We have also used heaps in our token management system.

Implicit bundle (virtual copies)

The implicit bundle consists in managing less tokens than the number of hypotheses they represent. A clear example is when the best hypothesis represent the set of competing hypotheses together. Since the ratio between the scores of these hypotheses is constant inside the lexicon network (as far as LM transitions are not applied), the actual scores of the hypotheses not explicitly represented can be recovered afterwards (this naive intuition is not as easy as we have explained, anyway). This technique, which remembers us the *V formation*[46] of some migratory birds, is very efficient during token propagation but not in the case of token recombination nor when a hypothesis reaches a word end and the LM transitions have to be incorporated. Let us briefly review how this and similar ideas have been reported in the literature:

- [Demuynck *et al.* 2000] describes a particular case of tokens representing several hypotheses which is applied only to homophone words. These words are emitted at the same time by the lexicon network decoder because they correspond to the same sub-word sequences. Instead of emitting all the words associated to this transcription, a special "super-word" is emitted. When a super-word reaches the end of the network and LM transitions are to be applied, it is split into individual tokens again;

45 As explained before, we do not agree with the use of the word "stack" and believe that "bundle" is more appropriate.

46 See `http://en.wikipedia.org/wiki/V_formation`.

- [Seide 2005] describes a technique for LM-conditioned decoders based on clustering the LM states in groups for which the optimal start time of words are identical. This same heuristic is described in the following point on word-pair and related approximations. Although a sole lexicon network decoder is used for each LM states group, each token inside this decoder is "virtual" meaning that it implicitly represents the set of tokens of all hypotheses which belong to the same group. The hypotheses associated to the individual LM states can be recovered afterwards. Note that the lexicon decoder instance performing the actual computation is working on upper bounds and , therefore, it is not equivalent to any of the lexicon decoders associated to the individual hypotheses represented by it.

1-best and word-pair approximations

The word-pair approximation [Schwartz and Austin 1991] is based on the observation that start times are usually similar for hypotheses that have their last word in common. Although these hypotheses are univocal when using a bigram LM, this property no longer holds in other LM types. Indeed, the word-pair approximation consists in using the same lexicon decoder instance for all hypotheses whose last word is the same no matter their actual LM state.

Observe that, when a given word is emitted from several lexicon decoder instances, only one of these hypotheses will survive at the root of the corresponding lexicon network instance, that is why this is an approximation.

Two generalizations of this idea are proposed in [Seide 2005]: on the one side, to cluster the instances based on the phonetic history rather than on its linguistic counterpart. On the other side, as described before, to create a "virtual" hypothesis that implicitly represents the set of hypotheses which are entered at the same lexicon network.

A more drastic technique is the 1-best approximation, described in [Kitaoka *et al.* 2005], where it is assumed that word boundaries do not depend on the past word. Although this approximation is very aggressive, its effect is diminished when using a linear lexicon instead of a tree lexicon as mentioned in Section 10.5.1.

10.6.7 Language Model Look-Ahead

Language model look-ahead (abbreviated LMLA) was introduced in Section 6.10.1 as a feature associated to language models which was related to decoding. It is a particular case of the capability for a LM to compute a (tight) upper bound on the probability of observing a word from a subset Ω' of the entire lexicon Ω:

$$\mathrm{LM}_{\max}(\Omega') = \max_{w \in \Omega'} p(w|h)$$

Although this value can be computed by brute force in any LM, more clever techniques can be used in some LM types. Let us now see how this feature can be useful for decoding in general and for the lexicon estimation in particular.

weight pushing covers LMLA

Note that this feature is not required in the static transducer composition approach because it is a particular case of weight pushing. LMLA makes sense in dynamic decoders where the lexicon decoder does not allow to know the identity of words until the final frame is processed. Some lexicon decoders, as those based on a linear lexicon (see Section 10.5), do not suffer from this problem. Unfortunately, the identity of words is not completely known in the most efficient lexicon decoders (e.g. the tree lexicon and the network lexicon). Active states in those decoders are associated to (sets of) prefixes which can be possibly completed in different ways, so that many active states can lead to more than one word. Let $W(s)$ denote the set of words which can be eventually emitted from a given state s.

The most obvious way to use those lexicon decoders in a dynamic decoder is to delay the inclusion of the LM score until the word identity is known. This contradicts the general and desirable advice of applying all sources of information as soon as possible. In particular, the problem of delaying the contribution of LM score is that hypotheses are differently over-estimated. For example, it is possible that a word with a high probability of being emitted, disregarding the LM, will be severely penalized by the LM, so that other words with a lower score, from the lexicon point of view, will become much better after applying the LM transition probability. A very aggressive pruning would discard those last hypotheses. There is also a mismatch between the hypotheses with and without the last LM contribution at the same frame, which is perfectly possible because different hypotheses can be associated to different word segmentations. An interesting discussion on when and how to include LM probabilities during decoding can be found in [Ravishankar 1996; Section 4.2.2].

LMLA alleviates this problem by advancing the LM contribution in the form of a conservative upper bound. The most obvious upper bound is the maximum LM score of all words which can be generated from the current active state s:

$$\max_{w \in W(s)} p(w|h)$$

This upper bound can be included in the transitions of the lexicon decoder in an incremental way (the difference of log-likelihoods from the previous LMLA contribution) and only needs to be stored in transitions from states with more than one successor. The extra cost required by LMLA leads to the following trade-off:

- on the one side, LMLA makes it possible to apply a stronger pruning to obtain the same error degradation, hence reducing the number of active hypotheses and, possibly, the decoding effort;
- on the other side, when the computation of LM transition probabilities can be very expensive. It may be better to avoid LMLA so that, hopefully, some lexicon decoder hypotheses would be pruned before requiring the computation of their LM score.

Relating the drawbacks, we can also mention that LMLA is more difficult to integrate in some decoders, as is the case of those used in two stage and in the time-conditioned search. This issue is addressed below. Moreover, some estimators proposed in this PhD manage several

LM hypotheses in the same lexicon network state. The inclusion of non-trivial LMLA would reduce the potential gains associated to this approach so that the benefits obtained by more sophisticated LMLA would not compensate a larger number of hypotheses processed per second.

The LMLA techniques can be extended to across-word decoders as described in [Sixtus 2003; Section. 7.2]. These decoders usually hypothesize several right contexts for the last sub-word unit of each lexicon entry. In this way, each of those hypotheses can apply some form of LMLA over the next word even if no frame has been processed. This "across-word LMLA" is based on the constrained vocabulary comprising words starting with the particular sub-word unit which has been hypothesized.

Unigram smearing

LMLA for free

In the previous discussion, when LM look-ups are very expensive, it is better to apply LMLA using a different, cheaper to compute, LM. This provisional LM score would be replaced by the true score afterwards. The most extreme situation is the use of a unigram for LMLA, a technique known as "unigram smearing" [Steinbiss *et al.* 1994; Alleva *et al.* 1996] which can be implemented at a really negligible cost because it is not required to distinguish different LM contexts. To implement this approach it suffices to properly modify the transition probabilities of lexicon decoders, with the unigram scores, during its construction. The approximated LM score has to be replaced by the correct LM transition probabilities.

unigram smearing can be transparent

An advantage of unigram smearing is that it can be made transparent from outside the network lexicon decoder: The linguistic or LM scores of the approximated unigram LM are taken into account by the lexicon network during decoding, and they are used exclusively to improve the performance of pruning techniques. When the word scores are emitted, the approximated LM information is removed. Note also that this approach can be used not only in LM-dependent one-pass decoders but also in semi-decoupled decoders.

This is the compromise solution we have chosen in our implementations, although other alternatives are pointed out below. This seems a reasonable choice since, according to [Huijbregts 2008; Section 8.1.3], an unigram LMLA is much faster than avoiding LMLA and is only 1.35 slower than a complete LMLA. We believe that our special methods for speeding up the lexicon decoders using cache oblivious algorithms would make this difference even smaller.

LMLA implementation for LM-conditioned decoders

Relating the use of LMLA in LM-dependent decoders, the LMLA upper bound can be included in the lexicon decoder in an incremental way in transitions from states with more than one successor when there exists a different copy of the lexicon decoder for each LM context. Instead of copying the entire lexicon model, it suffices to include a special index in those transitions (LMLA index) to access a vector storing the "compressed look-ahead tree". The same index can be used

to access the vector associated to each particular LM context so that each lexicon decoder copy only requires one of those vectors of scores [Ortmanns *et al.* 1996]. The problem of this approach is that it may require a lot of space when many different LM contexts are activated in a given moment.[47] These look-ahead vectors are cached[48] and created on demand by computing the values on the tree leaves and propagating (factorizing) them backwards to the root by means of a dynamic programming recursion or by sorting the nodes in topological and performing a reverse traversal. [Ortmanns *et al.* 1996] also proposes to consider only the first 2-4 arc generations of the lexical tree without any significant loss in accuracy.

It is possible to reduce the temporal and spatial cost of creating LMLA tables when using high order n-grams with back-off smoothing. To this end, [Nolden *et al.* 2011a] proposes to compute only an sparse table (fixed sized hash table) with the nodes which do not employ back-off and to perform backing off also when accessing these look-ahead tables.

sparse & backing-off

Let us now see how LMLA can be implemented in decoders using the per-stack organization (the use a sole lexicon tree decoder instead of the tree copies approach): In [Cardenal-Lopez *et al.* 2002] an small cache (hash table) is attached to each lexicon node to store its LMLA values. The decoder described in [Huijbregts 2008] is based on a sole tree lexicon decoder but, instead of a small cache at each tree node, it has a global hash table to access the compressed LMLA tables (as in [Ortmanns *et al.* 1996]). The improvement in this case comes from the fact that this table is only accessed by tokens entering the tree lexicon and a copy of a pointer to the corresponding LMLA table is kept in each token of the tree lexicon.

A trade-off between precision and computational cost can be obtained by selecting a different strategy depending on the number of word continuations of each state of the lexicon tree. This approach is proposed in [Massonie *et al.* 2005] where the chosen strategy varies depending on the number of word continuations: the exact computation of the LM score is computed when there are very few word continuations. The LMLA is approximated or even pre-computed when this value exceeds a threshold and, finally, when the list of word continuations is huge, the LMLA is skipped.

precision-cost tradeoff

This approach shows that the precision of LMLA values is not as critical as the LM scores themselves. Indeed, the effect of unigram smearing is quite remarkable. This make us think out the following technique we have not found in the literature: to pre-compute LMLA vectors[49] for each LM state in WFST (and n-grams represented by means of WFSTs) followed by a clustering procedure to limit their number to a predetermined size. Alternatively, we can fix a certain tolerance associated to the effect of pruning to reduce the number of different types of LMLA tables. It would suffice to store the index of the cor-

clustering LMLA vectors

47 Surprisingly, [Cardenal-Lopez *et al.* 2002] reports a very low value of 10-30 different LM histories per frame.

48 According to [Cardenal-Lopez *et al.* 2002], caching reduces the effort of computing these tables in a factor of ten.

49 The internal representation of those vectors could profit the ideas of [Nolden *et al.* 2011a] to reduce the size and cost of the computation in case of back-off models.

responding type of LMLA associated to each different LM state. This is expected to improve the unigram smearing and, indeed, resembles the approach applied in some decoders based on the bigram smearing approximation where the number of LMLA vectors is the lexicon size and can be pre-computed in some cases.

Finally, let us remember that lexicon-free LMs based on n-grams with a high order can be used to obtain LMLA in a very efficient way without requiring to compute the set of LM scores for all words from the vocabulary. Similarly, NN LMs can be designed to compute these values associating a different softmax layer to each lexicon state with a fan-out greater than one as in the "LMLA NNLMs" proposed in Section 9.5.4.

LMLA for 2-stage and time-conditioned decoders

The use of LMLA (beyond unigram smearing) seems more difficult to implement in the time-conditioned search approach because hypotheses in each lexicon instance share the starting time and, when emitted at the word level, can be used in several LM contexts. In this case, the upper bound is no longer dependent on the possible word continuations for a sole LM context, but it has to be computed for a set of LM contexts. The obvious solution consists in extending the idea of upper bound to include the bounds provided by the different LM contexts. Quoting one of the few references to this issue [Nolden *et al.* 2010; Section 2.1]:

> *Such a high effort does not pay out*

Relating this issue, we would like to point out that the implementation of LMLA in this type of decoders is more straightforward with a novel feature proposed in this work. The crux of the matter is that time-conditioned search is essentially equivalent to the semi-decoupled decoders described in Chapter 8 and we can, therefore, take profit of the feedback channel from the DAG decoder back to the DAG generator. This feedback has been used in our experiments to provide bounds suitable for performing the same kind of pruning than we are seeking here and that, in more coupled classical decoders, require an ad hoc implementation.[50]

In order to add LMLA in this technique, it suffices to include different ways to perform a transparent LMLA in the set of pre-determined lexicons (from a given cluster as described before) where, by "transparent" we mean the use of anticipated LM probabilities which are removed when emitting the tokens from the DAG generator to the DAG decoder. The most appropriate type of decoder, not only w.r.t. the set of recognized words but also w.r.t. the LMLA, can be chosen *on the fly* at the beginning of each frame boundary based on the actual use that will have the produced tokens in the future. This information can be efficiently computed (in a cost linear with the number of LM contexts). This feedback feature is described in Sections 11.1.3 and 12.2.

50 Relating this feedback, let us remark that some additional information can be sent in the form of sub-lexicons indices which can be used to select a tighter lexicon decoder. This extension has been described in Section 9.1.2 (concretely, in the sub-section titled "Working with pre-determined vocabularies").

Related alternatives

A technique related to LMLA, described in [Yu *et al.* 2014; Section 6.1], called "LM slack in pruning" by theirs authors is used together with LMLA to to smooth over time the effect of adding LM scores and, hence, prevent over-pruning when the rest of the LM score is included at word-ends. The technique consists in including a new variable in the tokens called slack variable. This variable is multiplied by a penalty at each Viterbi step in order to make it decrease exponentially. The LM scores are also accumulated in this variable which is also taken into account by the beam search.

An heuristic resembling LMLA is described in [Kato *et al.* 2010]: Pruning hypotheses close to the root have a worse impact than pruning the ones closer to the leaves. Although this is not LMLA, its purpose is similar and can replace or be used in combination with it. This heuristic can be easily implemented by including a reward depending on the number of words reachable from each state in the pruning stage. Since this reward does not depend on the LM context, we wonder which is the relationship with unigram smearing.

Another technique close to LMLA is to prune hypotheses at the beginning of the lexicon decoder by taking into account the best LM score associated the corresponding LM context, which seems a particular case of LMLA applied in the initial lexicon decoder states. Note that the LM interface described in Section 9.1 contains a method to get the best LM score reachable from a given LM history, which is used to implement this particular pruning technique.

10.6.8 Submodel dominance recombination

Submodel (or sub-tree) dominance recombination [Ortmanns *et al.* 1998] is a pruning technique that can be applied to one pass decoders where hypotheses are characterized by their state in the lexicon network and by their LM context (e.g. LM-dependent and re-entrant or token passing decoders). This technique allows to safely remove (i.e. without sacrificing optimality) some hypotheses thanks to other ones sharing the same lexicon state but with *different LM contexts*.

To safely remove the hypothesis h, we have to guarantee that any hypothesis obtained from h will be discarded in the future before the end of the sentence. In particular, let us see a condition to determine that they will be discarded when the lexicon emits the next word. To this end, we only have to observe: 1) that all hypothesis associated to the same lexicon network state s will increase their score *by the same amount*, excepting the LM contribution, and 2) that the set of words achievable from s, denoted by $W(s)$, is also the same.

From these considerations, in order to determine that h can be removed, it suffices to see what happens with h, w.r.t. other hypothesis associated to s, for each possible word from $W(s)$. In other words, h can be removed if $\forall w \in W(\text{state}(h))$ there exists an hypothesis h', $\text{state}(h') = \text{state}(h)$ such that:

$$\text{score}(h)p(w|\text{LM}(h)) < \text{score}(h')p(w|\text{LM}(h'))$$

Since this checking is very costly, a simplification is usually performed by taking upper and lower bounds. It suffices to compute, for each state s and each LM context v, the best and worst LM contributions:

$$\text{upper}(s, v) = \max_{w \in W(s)} p(w|h), \ \text{lower}(s, v) = \min_{w \in W(s)} p(w|h)$$

In this way, we can group all active states associated to s, combine their current score with the corresponding lower bound and keep the best combination. All states different from the one that won the previous maximization can be removed if their score, combined with their upper bound is worse than the previous value. Note that the upper bound is the same value that is usually computed for LMLA. This technique seems simpler to implement in the per-stack organization where all the hypotheses associated to the same lexicon network state are kept together. Although this technique has not been implemented in our toolkit for the moment, some of the lexicon decoders proposed in this work are based on a double ordering to implement the bundle search from Section 10.6.6. This particularity makes it easier to include this pruning technique.

10.6.9 Other techniques to reduce the cost

Although each of the following techniques might probably deserve a proper section, they are not as extended as previous ones since they cannot be applied in general decoders. Some techniques rely on the use of oversegmenters, on the use of representations without sets of active states or on special particularities of the input data.

Jointly decoding several frames

The idea of jointly decoding several frames consists in performing several Viterbi steps *in the same traversal of active states* by assuming that no tokens will arrive to the lexicon decoder in the during a predetermined number of consecutive stages. A possible scenario where this feature can be used is in the lexicon networks of DAG generators or in one-pass decoders based on a re-entrant network when used with an over-segmenter. The constraints provided by such over-segmenter would allow us to know that no incoming tokens will appear in the re-entrant network at certain frame boundaries.

The technique seems novel, to the best of our knowledge (we have not found it through the extensive literature) and a probable reason for that is that over-segmenters are not used in HMMs decoders either. However, the use of over-segmenters is thoroughly discussed in this work (in this chapter and in Chapters 8,11 and 12) for other purposes as well. Although over-segmenters are not reliable in ASR tasks, we have experimentally shown that they can be used in some HTR tasks, as reported in Chapters 14 and 15.

In order to make the proposed approach simpler, we also need to assume that, besides the use of over-segmentation, no intermediate hypotheses emitted by one state will ever arrive at its input during the frame sequence that is jointly processed. This last assumption can be easily relaxed to allow self loops. A notable family of HMM topologies

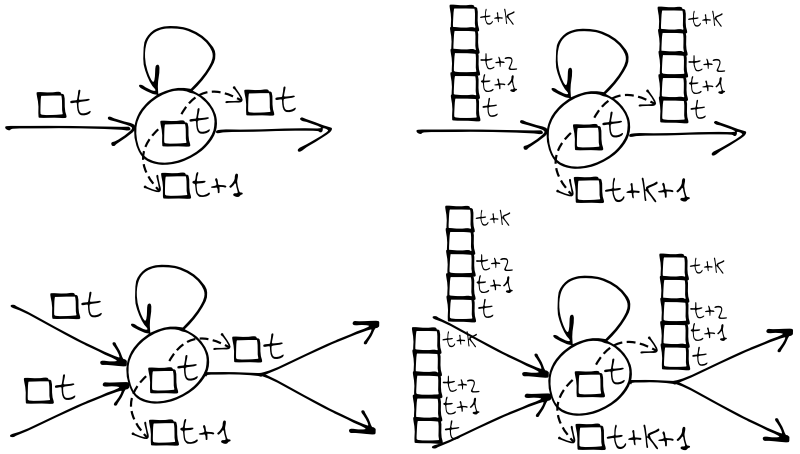


Figure 137: Jointly decoding several frames. In the left column: the classical update of the active state of a senone model during a Viterbi step. The scores of the incoming states at time t are used, together with the emission probability (not depicted), to update the score at $t+1$. In the right column we can observe the underlying idea of jointly decoding several frames: the scores received from the preceding states contains the scores of several consecutive instants $t, t+1, \ldots, t+K$. The state only contains the score at time t, but their corresponding scores at times $t+1, \ldots, t+K+1$ can be locally computed from the input vectors received by the incoming transitions together with the vector of emission probabilities at those instants (not depicted either). A similar set of tokens, corresponding to the instants $t, t+1, \ldots, t+K$ is propagated by the outgoing transitions.

satisfying these restrictions is the case of acyclic networks of senone models (Section 7.3.1). Fortunately, this leaves a plethora of practical topologies to our disposal, since there exists several lexicon network topologies that can be described in this way, as is the case of tree lexicons where the underlying sub-word models, based on left-to-right HMMs, have been expanded.

So, let us briefly illustrate how this technique works and, in particular, let us focus our attention on a senone state s of one such network. Assuming that s is active at time t, it suffices to know the hypotheses of the preceding states and its corresponding observable emission likelihood in order to update its score at time $t+1$, as illustrated in Figure 137*(upper left)*. The updated score can be computed as follows:

$$\text{score}(s, t+1) = \text{emiss}(s, t+1) \max_{s' \in \text{pred}(s)} \left\{ p(s'|s)\, \text{score}(s', t) \right\}$$

If we receive, from each preceding state, not just the token at time t but also the tokens at subsequent frame boundaries $t+1, \ldots, t+K$, we can compute the updated scores of state s at these instants, as illustrated in the right column of Figure 137.

In this way, the intermediate stages are sent from one state to its successors but the active state representation only needs to store the last computed score. Relating final states (those where we can emit word labels), they are able to emit words associated to these intermediate frame positions without problem.

Several particularities have to be taken into account for this algorithm to work properly: let us remark that new states can be activated not just in the current instant t but also at any of the intermediate instants in the range up to $t + K$. This means that a careful strategy is required. An algorithm which is able to deal with this issue is based on traversing active states in the topological order of the network since, in this case, a vector of tokens associated to several consecutive time positions can activate the following state and this new state, in turn, can activate the following ones and so on. This chain of activations in sequence is not produced when processing just one frame at each Viterbi step but, as will be illustrated in following Sections, may occur when dealing with error correction edition operations (and, particularly, with deletion). Let us remark that the traversal of active states in the topological order is just the contrary of the piggy-backed technique described in Section 10.3.3. Indeed, both the reversed and non-reversed traversals can be applied in two highly related specialized algorithms for linear and tree lexicon network topologies, the forward traversal versions described in Sections 10.8 and 10.9. The forward traversal can easily include the jointly processing of several frames described in this section.

reduce data traversals

The biggest advantage of the proposed technique is to reduce the number of data traversals and to avoid the construction of active states representations for the intermediate steps. Relating the drawbacks, it seems slightly more complex to incorporate pruning techniques.

Dealing with repetitions and using compression

In many tasks, such as ASR and HTR, it is practically impossible to find common sub-strings and, consequently, repetitions in the input data. The reason is that even the same user pronouncing or writing the same words would produce each time a different outcome (represented by means of continuous frames). In other tasks also related to sequences with a finite alphabet of discrete symbols, as is the case of some bioinformatic tasks, it is more reasonable to find common repeated sub-strings. There exists several works which address the idea of taking profit of these repetitions to speed-up decoding with HMMs: [Mozes *et al.* 2007] [Weimann 1997; Chapter 2] [Nielsen and Sand 2011] [Sand *et al.* 2013]. The rationale of these techniques seems to be the possibility of expressing the computation of HMM related algorithms as the product of matrices, as explained at the beginning of Section 10.2.1 where this concept was used to explain that segment estimation could be performed in several orderings, including the parallel reduction technique described in [Nielsen and Sand 2011] and illustrated in Figure 121. From the matrix multiplication point of view, it is clear that the computation associated to common sub-strings can be reused and is also obvious that this acceleration can be performed in a hierarchical way when there exists sub-strings appearing in other repeated sub-strings. These approaches are more suitable for HMM topologies with a reduced number of states, scores are computed for all states and there are chances to find repeated sub-strings, which is not the case of most of the tasks we have at mind.

Reducing the average cost of the exact Viterbi computation

An interesting technique to improve the cost of the Viterbi algorithm was proposed by Patel in [Patel 1995]. The technique is valid for fully connected models when not distinguishing a subset of active states. The standard Viterbi algorithm for computing a Viterbi step in these models has an asymptotic cost of $O(N^2)$, being N the number of states. This seems reasonable when taking into account that the number of edges is also $O(N^2)$ and that all the edges, in principle, make part of the expression to be computed:

$$\text{score}(s, t+1) = \max_{s' \in \text{pred}(s)} \{p(s'|s)\, \text{score}(s', t)\}$$

The quadratic cost comes from the fact that there are N scores to be computed and each one is a maximization over N products (or summations, when using logarithms). The idea described in [Patel 1995], allowing a reduction of the average asymptotic cost to $O(N\sqrt{N})$, is based on processing first the addends of highest score when computing the maximum of a set of summations so that, later, summations involving lowest scored ones can be safely skipped. This kind of tricks are also used in some shortest path algorithms.

A first preprocessing step consisting in sorting the scores of each set $\{p(s'|s), 1 \leqslant s' \leqslant N\}, \forall 1 \leqslant s \leqslant N$ is performed ahead of time and. It should also be possible to retrieve the index s' associated to each score.

Another operation is required before starting the following general algorithm applied over each destination state: to select the best k scores of the states at time t. Although these scores are not sorted among them, they are greater than (or equal to) the remaining $N - k$ scores. The the cost of this operation, known as selection, is $O(N)$. Although this operation cannot be computed ahead of time, it is computed once per each Viterbi-step.

After these pre-processing, we can compute, for each destination state s, its updated score using the aggregation[51] approach of Viterbi. However, we split the process into two phases:

1. compute score(s,t+1) using only the k highest scored states selected before. A particularity of this first maximization is that each time a score $p(s'|s)$ is used, the highest index of the corresponding sorted vector is updated in a variable called **MaxRank**;
2. now, the important insight is that only the transitions $p(s'|s)$ whose indices are above **MaxRank** can improve the partial maximization. We only need to use the values stored in $p(s'|s)$ from **MaxRank** to the end and picking the corresponding origin state (from the set of $N - k$ states not used in the first phase) updating the value $\text{score}(s, t+1)$ accordingly.

The first phase involves evaluating and maximizing k addends whereas the second one requires N−**MaxRank**, which depends on the actual value of **MaxRank**. It can be proved that taking $k = \sqrt{N}$, the expected value $E[\textbf{MaxRank}]$ is $O(\sqrt{N})$.

51 The distinction between propagation and aggregation was described in Section 10.3.

Although this technique can be extended from fully connected to more sparse models [Patel 1997], it is not evident to combine with the idea of using explicit sets of active states which are, in general, based on the propagation version of Viterbi. Active states with pruning seem a better approach for many decoding tasks, making the technique described in this section restricted to particular niches.

10.7 DETECTING AND EMITTING OOV WORDS

Lexicon decoders have a predefined lexicon but, in most real tasks, a closed lexicon is not feasible. OOV words can be handled at different levels. This issue has been discussed in Section 6.3.1 at the LM level: The language model can estimate the probability of a OOV word but, unless the lexicon estimator is able to emit hypothesis associated to OOVs, these LM features are usually untapped.

The capability of lexicon estimators to detect OOVs is not common. Some approaches to deal with the lack of this feature include:

- ignore the problem at the lexicon estimation level. It is possible, for instance, to obtain confidence measures at the output to detect where some OOVs can be produced. It is based on the assumption that OOVs are present in badly recognized regions. We do not like this approach since the presence of OOV is just an heuristic applied where it is too late to act more properly. A decoder ignoring OOVs makes the search procedure to follow wrong paths (probably guided by a wrong LM history) and this usually causes the mis-recognition of neighboring words;

too late!

- combine the lexicon estimation together with the estimation of sub-words. This approach, known as "hybrid LMs", was described in Section 6.8. There exists two different methods to combine the lexicon and the sub-word models, as illustrated in Figure 86: in a flat and in a hierarchical way.

drawbacks of hierarchical hybrid LMs

Let us remark that the "hybrid LM" has some drawbacks due to the fact that: 1) the lexicon-free model to detect OOVs may perfectly recognize words from the lexicon, and 2) conversely, the lexicon decoder may assign a reasonable probability to some words when pronouncing a OOV. This is the reason why a careful weighting of both models is required. Observe that this solution is not the same as having a lexicon decoder which may emit OOVs (labeled with the <UNK> symbol).

The technique that we propose for detecting OOVs in the lexicon decoder (and independently from the LM and from the global decoder) has some resemblances (despite being different) with hierarchical hybrid LMs (Figure 86*a*)). Its rationale is based on the following heuristic:

> *A lexicon free decoder is expected to give better scores than the lexicon based decoder counterpart when dealing with out of vocabulary words since the lexicon based decoder has to match the input evidence with an approximate but incorrect sub-word sequence. However, the lexicon free model may use the correct one.*

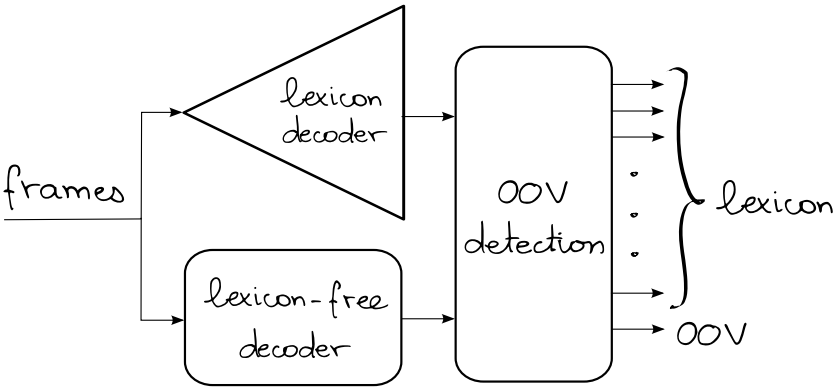


Figure 138: Decoder able to output OOV probability by combining the result of two decoders working in parallel, one with a lexicon an other lexicon free. OOV detection is based on several heuristics: when OOV words appear, the difference of scores between lexicon decoder and lexicon free decoder increases, the Levenshtein distance between the N-best outputs of the lexicon-free model and the closed lexicon increases too.

This can be implemented by including an additional module which takes into account the set of scores produced from the internal lexicon decoder and lexicon free parts, as illustrated in Figure 138.

This module is responsible for properly deriving an emission probability of the OOVs word by taking into account not only the scores of each part but also the particular sets[52] of sub-word models recognized by the lexicon free sub-model. The information gathered by the integration module can be used with one of the following techniques:

- the Levenshtein distance between the N-best sub-word sequences produced by the lexicon-free decoder and our lexicon. This would discard the output of lexicon free decoder associated to known words. Another approach is to explicitly forbid lexicon words from the lexicon free decoder by constructing an ad hoc model based on the complementary of our lexicon. This would lead to a much more complex model. The main advantage of a complementary model is that it could be used in LM-conditioned one-pass decoders;
- the normalization of the set of likelihoods of words associated the observed segment may be considered as an approximation of posteriors which can be used to estimate the probability of OOVs by measuring, for instance, the entropy.

Note that these techniques can also be used to estimate the reliability of the lexicon decoder in some parts of the recognition. This information could be used as input for adaptive recognition techniques.

It remains to determine and how to properly adapt the overall system in order to associate the words recognized with the <UNK> symbol with the corresponding transcriptions produced by the lexicon free recognizer.[53]

52 Not only the 1-best but the N-best paths.

53 In case they are required. It is not obvious how to produce them in certain tasks.

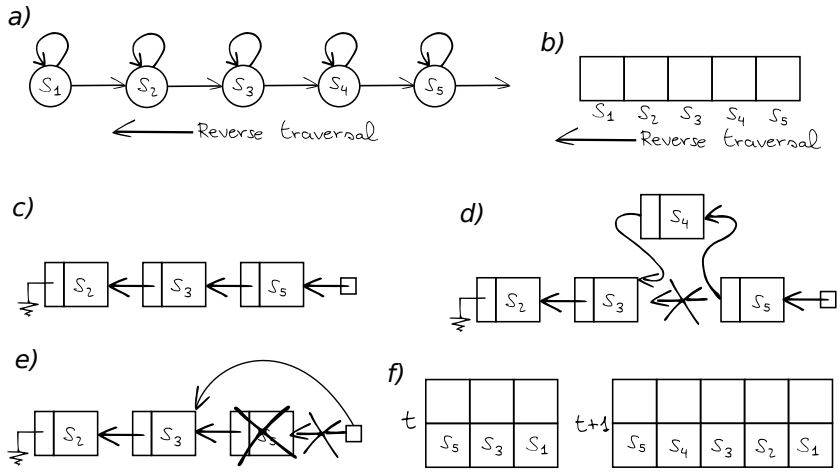


Figure 139: Some issues related with applying the reverse traversal version of Viterbi algorithm to a linear sequence of senone models: a) illustration of the reverse traversal of states in a linear lexicon topology, b) the reverse traversal when using an array representation and when a subset of active states is not distinguished, c) alternative linked list representation suitable for explicitly representing active states, d) new states can be activated in place when implementing the destructive or persistent version of the Viterbi-step, e) how states can be pruned or deleted in the same persistent scenario, f) using an array explicitly representing active states and persistent operations. Observe that in this last case the updated set of active states is obtained in a new vector and it is possible to store and to perform forward and backwards traversals.

10.8 SPECIALIZED LINEAR LEXICON DECODERS

Our main aim is to obtain specialized tree-lexicon and network-lexicon decoders, but we believe that it is better (for didactic purposes) to start by explaining the same ideas applied to a linear or flat lexicon because many conclusions and basic ideas can also be drawn from these simpler models. We hope that, in this way, the proposed designs would seem as obvious as possible.[54] We will first suppose that these linear lexicons are sequences of senones (see Section 7.3.1) as illustrated in Figure 139*a)*, although they are later extended to use other more complex topologies. These techniques will be first extended from linear to trees and later from trees to networks, respectively, in the following sections. Nevertheless, the algorithms developed here for linear lexicons are not limited to didactic purposes: they may be used in practical decoders, particularly when used as sub-word models together with a technique that, since we have not seen elsewhere in the literature, we have coined as "sub-word centric lexicon decoding".

54 It would be possibly more convenient for us just to describe the more elaborated designs and "try to muddy the water to make it seem deep". But we rather believe that the best proof that an idea is well structured and well transmitted (an additional and necessary iteration over the invention) consists in providing the necessary introduction to make it seem a dumb combination of prior art, even if the point of views making that so dumb were unknown to the author of the idea in the moment of the invention. Indeed, these points of view are often a worked sub-product of first elaborating the idea.

As explained in Section 10.3, there exists several possibilities when implementing even the simplest variations of Viterbi algorithm:

- it is possible to choose between two different modalities: propagation and aggregation. In the first one, the outgoing transitions are traversed. In aggregation, destination states are computed one at a time by using their incoming transitions;
- are we explicitly representing a subset of active states? which proportion of states have a non negligible score?
- in which order are states traversed? some particular orders allow, for acyclic networks, an in-place update;
- how are states represented? several choices exists such as arrays, hash tables, etc. This is particularly important when explicitly representing active states;
- are back-pointers required to retrieve the best path?

Other question addressed in Section 10.2.2 is the use of persistent or ephemeral operations, which mainly depends on the topology of the input DAG (meaning that persistent operations are basically limited to frame sequences and, in a possible extension, to confusion networks).

These choices are not orthogonal but highly interrelated since some combinations are more natural while others seem cumbersome. There are also efficiency issues: it is important a fast traversal, good patterns of memory accesses and a fast look-up when looking for active states.

Dense representation, left-to-right or right-to-left?

Although linear lexicons made of more complex sub-word model topologies will be addressed later, let us first see how these issues are related with the particularities of linear lexicons made of senone units: The most obvious property, *when the self loop transitions of senones are not taken into account*, is that they constitute a particular case of an acyclic model. As already mentioned, this allows a reverse topological traversal of states for an in-place updating, known as piggy-backed technique, as described in Section 10.3.3.

This algorithm, when applied to linear lexicons, can take profit of the fact that each state, excepting the initial one, has just one predecessor. The resulting pseudo-code is illustrated in Algorithm 8, where v contains scores (e.g. scaled log-likelihoods) and is indexed from 0 to $\mathtt{num\_states} - 1$. Transition probabilities are modeled with $\mathtt{loop}$ and $\mathtt{itrans}$ vectors (the last one containing *incoming* transitions). Emission likelihoods are pre-computed and stored in $\mathtt{emiss}$ vector.

Data: score $\mathtt{entry\_scr}$, vector v of size $\mathtt{num\_states}$
Result: v is in-place updated
for $i \leftarrow \mathtt{num\_states} - 1$ **down to** 1 **do**
  $v[i] \leftarrow \max(v[i] + \mathtt{loop}[i], v[i-1] + \mathtt{itran}[i]) + \mathtt{emiss}[i]$
$v[0] \leftarrow \max(v[0] + \mathtt{loop}[0], \mathtt{entry\_scr} + \mathtt{itran}[0]) + \mathtt{emiss}[0]$

Algorithm 8: Piggy-backed Viterbi specialized for linear lexicons.

forward & backwards traversal

Observe also that, since each state has one successor, it is also easy to implement an in-place updating Viterbi-step using a left-to-right ordering and traversal of states provided an auxiliary variable is used to store the current state (score and token) before modifying it. Otherwise stated, both right-to-left and left-to-right traversals are possible. The reader may wonder which is the point of this slightly more complex version: forward traversal allows the inclusion of certain features such as skips made of null transitions, error correction edition operations or the use of the technique based on jointly decoding several frames proposed in Section 10.6.9.

Active states, lists and arrays

Although all these variations can be described for the case of dense representations where all states are explicitly represented and manipulated, we will focus our attention, in what follows, in the use of an explicit subset of active states. It seems quite obvious that this kind of Viterbi decoders are restricted to the propagation approach since the potential gains would be lost when using aggregation. Moreover, the techniques described in Section 10.3 for these cases made use of hashing or similar techniques to determine whether or not a destination state has been activated and, in that case, to locate its previous value.

The specialized algorithms described in this section will take profit of particular features of linear lexicons to improve this baseline by increasing memory locality, reducing the cost of active state look-ups and, hence, allowing faster implementations.[55] The main feature to consider is:

> *If the set of active states is contiguously stored sorted by the state identifier, the set of states reachable from them is also generated in a sorted way.*

We will see that a similar property holds for tree lexicons, when using certain sorting of states, but not for the more general case of DAWGs.

Arrays or linked lists?

Linked lists seem a convenient way to explicitly store active states. They can be efficiently traversed and it is trivial to determine, depending the right-to-left (as shown in Figure 139*c)*) or left-to-right ordering, if the origin or the destination of an active state is also active: it suffices to check if the current node is the last one or, in other case, to check the state identifier of the next node of the list. When using the right-to-left ordering, we can easily activate the destination state (as illustrated in Figure 139*d)*) or remove the current state if it is pruned (as depicted in Figure 139*e)*).

When using an array, both right-to-left and left-to-right active state traversals are reasonable choices but, contrarily to the dense representations, a secondary array is now required to store the result, as illustrated in Algorithm 9.

The implementation of Algorithm 9 is easier than the corresponding linked-list approach due to the fact that we are not performing an

55 Meaning that less cycles are used when processing each active state. This can be measured as hypotheses processed per second (cpu usage or wall-time, depending).

```
Data: score entry_scr, score threshold, vector v and its size sz
Result: vector v' and its size sz'
sz' ← 0
prev_st ← 0
prev_scr ← entry_scr
for i ← 0 to sz − 1 do
    st ← v[i].st
    if prev_st + 1 < st then
        scr ← prev_scr + itran[prev_st + 1] + emiss[prev_st + 1]
        if scr ⩾ threshold then
            v'[sz'].st ← prev_st + 1
            v'[sz'].scr ← scr
            sz' ← sz' + 1
        scr ← v[i].scr + loop[st] + emiss[st]
    else
        scr ← max(v[i].scr + loop[st], prev_scr + itran[st])
              +emiss[st]
    if scr ⩾ threshold then
        v'[sz'].st ← st
        v'[sz'].scr ← scr
        sz' ← sz' + 1
    prev_st ← st
    prev_scr ← v[i].scr
if prev_st + 1 < num_states then
    scr ← prev_scr + itran[prev_st + 1] + emiss[prev_st + 1]
    if scr ⩾ threshold then
        v'[sz'].st ← prev_st + 1
        v'[sz'].scr ← scr
        sz' ← sz' + 1
```

Algorithm 9: Forward Viterbi for a senone sequence using active states stored in an array.

in-place updating.[56] Nevertheless, the resulting code is slightly complicated due to the fact that states can be activated and deactivated due to pruning. Although a naive pruning threshold is shown, more complex pruning strategies, as described in Section 10.6.1, would be used in practice. An alternative approach to implement this algorithm would be to generate an intermediate array to deal with loops and another one to deal with outgoing transitions in order to finally merge both of them, as illustrated in Figure 140. Although this alternative version is not as efficient, it clearly explains the fact that the order is preserved when applying the loops and the outgoing transitions, respectively, and the fact that two lists containing sorted active states can easily be merged. Algorithm 9 can be seen as a specialized version without these intermediate data structures.

56 It would be pointless to implement a persistent version with the linked list representation. Persistent linked lists make sense when list tails are probably shared, which is not the case in our scenario.

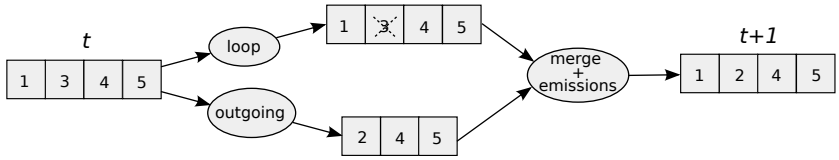
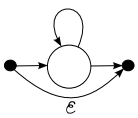


Figure 140: Scheme of an algorithm for linear sequence of senones using arrays. Although not specially efficient, it clearly shows the idea that loops and outgoing transitions preserve the order so that the results can be merged. The model is assumed to have 5 states, which explains why state 6 is not generated during the "outgoing" process. State 3 is removed after during loop to illustrate the possibility of pruning.

A *preliminary* conclusion when comparing linked lists and contiguous arrays is that the usefulness of linked lists seems restricted to the ephemeral, destructive or in-place updating modality, while the sparse array representation is more suitable for the persistent version since a new copy of active states is obtained. Relating the spatial cost, the storage requirements per state are increased w.r.t. a dense array because an index of the HMM state is required per state, but linked lists require additionally a pointer to the next element. Back-pointers are not considered in the examples although we would limit to move them around as (dummy values in the tokens) since we are not interested in the alignment at the fine grain level of sub-word or HMM state.

However, let us remark that nothing prevents us from using the persistent versions when the ephemeral one would suffice. Indeed, one of the simplest implementation strategies consists in using two different arrays that are interleaved for t and $t+1$ as illustrated in Figure 128b. We believe that arrays are more convenient than linked lists for implementing the classical ephemeral version of Viterbi-step for at least four reasons: 1) the code for traversal is faster, 2) memory accesses are more cache friendly, 3) they occupy less memory, and 4) the same representation is valid for both forward and backwards traversals (whereas single-linked lists have to choose the suitable direction). Let us see how these ideas can be extended to scenarios more complex than simple senone sequences in order to extend them later to tree lexicon topologies.

Extension to models with null skips



The presence of null skips is quite similar to the case of deletion operations of error correcting decoding described in Section 10.4. One of the techniques described in that section to deal with deletions took profit of the acyclic property of states (again, loops can be tackled as a special case) to perform a topological traversal. In our current linear case, this is straightforward since it suffices to maintain the forward traversal of Algorithm 9 and not only check v but also v' that is being constructed. The only trick in that case is that, in principle, an indefinite number of states can be activated from a given one.[57]

57 When creating $v'[sz']$ it is possible to activate $v'[sz'+1]$ and so on. We have to proceed in this way (in a loop to be applied each time a new state is created) until one of the

Dynamic expansion of sub-word models

Typical lexicon decoders consider a repertoire of sub-word units (phonemes, graphemes,...) and a lexicon made of sequences of these sub-word models. When the lexicon representation is flat and sub-word models are linear left-to-right senone sequences, the resulting expanded model is also a linear senone sequence, but it would be a waste of space to statically expand it.

By "dynamic expansion" we mean that the decoder is able to use a representation where sub-word models are not expanded or replicated but shared in a common dictionary. On the other hand, the lexicon is stored as sequences of sub-word model indices. Several strategies can be envisaged when implementing this approach:

- the entire sub-word unit is completely activated or deactivated. This means that, when it is activated, a dense representation for this particular sub-word model is managed. This may simplify the code and is reasonable when these sub-word units have a reduced number of states and most probably only a few proportion of sub-word units is active. Just to give some numbers, typical ASR left-to-right models have only three states. However, HTR models may typically have 7 or more states, making this approach less attractive;
- HMM states can be independently activated from others of the same sub-word unit as if the model was statically expanded. This case is particularly easy when sub-word models are linear senone sequences and, in that case, the only remaining difficulty is how to encode the state identifier[58] and to check whether or not the following active state belongs to the same sub-word unit, if it is the last one and so on;
- a third alternative, that we have not seen in the literature, is to maintain a contiguous subset of active states. The advantage of this approach it that we space required to maintain the indices of active states is reduced to just two values not matter the actual size of the model or the proportion of active states. This approach seems a balance between the previous cases. Although we have not empirically validated the suitability of this approach, prior experience with a related task (Viterbi forced alignment) has shown that it is reasonable in practice.

Sub-word units with complex topologies

Previous examples have used linear sequences of senones where each one receives a transition from the previous one. Let us describe different ways to increase the complexity in the topology:

three following cases applies: 1) pruning deactivates the created state, 2) the last state is achieved, or 3) we reach the state to be created from v.

58 When all sub-word units have the same number of states, modular arithmetic may suffice to extract the lexicon and sub-word state indices from a combined value. In other cases, we can pack the sub-word HMM state and the sub-word id in the same word identifier in other ways (e.g. by taking an upper bound on the number of HMM states. If numerical ranges allows it, a power of two values convert this modular arithmetic into a simpler bit manipulation).

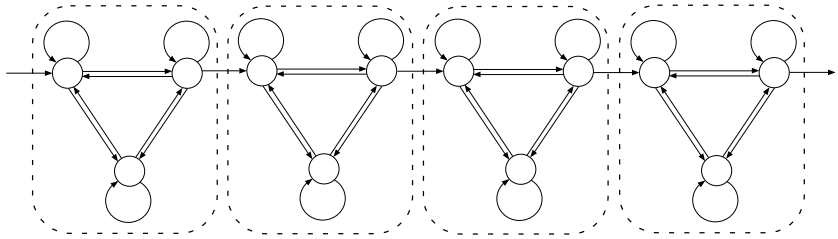


Figure 141: Linear lexicon with complex topologies. In this example, each sub-word unit is an 3-state ergodic model.

- by connecting non-adjacent models;
- by replacing senones with more complex models;
- by increasing the number of entry and exit points.

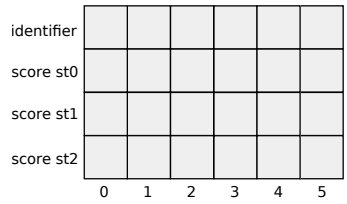
Bakis HMMs

In Bakis HMMs, described in Section 7.3.1, each state is not only connected to itself and to the next one but also to the next of the next, as illustrated in Figure 92*c)*. It is quite easy to adapt Viterbi for this topology in both forward and backwards traversals for both dense and active states representation (either in the form of arrays or linked lists, as well). In the case of active states representation where states are sorted by the state identifier, we had to check the identifier of the next active state and to check if it is either the next state or the next of the next state. Only when it is the next state, we have to check the next of the next active state. Using intermediate arrays (as in Figure 140) we would need a third intermediate array and to perform a 3-way merge while the specialized version would be more complex than Algorithm 9.

complex sub-word units

The use of a linear topology made of parts more complex than a senone sequence is illustrated in Figure 141. In that example the sequence is made of 3-state ergodic models. In that case, the reasonable way to represent them is by using the aforementioned dynamic expansion.

Figure 142: A possible data organization for representing linear sequences of complex sub-word units. An array of a structure with an identifier and a sub-array of scores is used to represent models as those of Figure 141 (other data organizations can be envisaged).



In the simpler case when all active models have the same number of states and are jointly activated, we can use a vector of structures where each position contains the information associated to each activated sub-word unit. An example of such a representation for the model of Figure 141 is shown in Figure 142. Each of these components can be managed as if it was a complex senone model leading to algorithms close to that of Algorithm 9 where the management of the scores of the sub-word unit HMM states is managed by a (possibly unrolled) internal loop.

However, a more flexible approach that allows the use of sub-word units of heterogeneous topologies (and different number of states) consists in allocating first the sub-word identifier followed by a variable length number of positions depending on the actual stored model. Each different sub-word model is responsible of determining the actual size and to provide the outgoing score to the next sub-word unit. It is even possible that some sub-word units use a dense representation while others may use a sparse one.

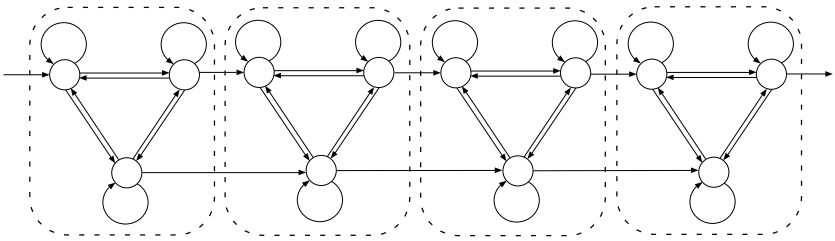


Figure 143: Linear lexicon with complex topologies and several entry points.

Besides the possibility of connecting each of these models not only to the next one but also to a reduced number of non-adjacent models (as in Bakis topology), we can also consider to allow more than one entry and exit points. An example of this feature would be to allow the connection of more than one state of the ergodic models of Figure 141, as shown in Figure 143. These features remember us the dynamic graphical models (DGMs) described in Section 4.1.1 where the ground model can be constructed for a given length by unrolling a time-slice or chunk template.

Managing multiple hypotheses: double ordering and merging

One of the unique features of the decoders proposed in this PhD is the double ordering of active states: They are first sorted following the lexicon network state identifier and, secondly, using the token identifier used to distinguish non-competing tokens in the same network state. This situation may arise at least in the following cases:

- in LM-conditioned one-pass decoders, several lexicon decoder instances are active, each one associated to a different LM context.[59] Tokens associated to the same lexicon state are sorted by their LM identifier. In that case, a sole lexicon decoder instance may be active and all active states associated to the same lexicon decoder state may be tackled together;
- similarly, in a time-conditioned decoder, several lexicon decoders may be active, each one associated to a different time-start. This is related to the DAG generator described in Section 11.1. In that case, the set of hypotheses associated to the same lexicon HMM states differ in and are sorted by the time-start value.

59 In the particular case of linear lexicons for bigrams, each word instance has the same unique LM context as indicated in Section 10.5.1 in Figure 132a).

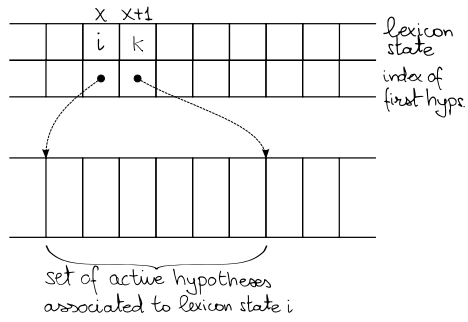


Figure 144: Data organization for managing multiple LM hypotheses at the same time. The active lexicon states are sorted by their state identifier and the LM hypothesis associated to the same lexicon state are sorted by their LM identifier.

The idea of maintaining a sorted list of active states of Algorithm 9 is extended to the double ordering by converting the operations over a sole active state into operations on sets of hypotheses associated to the same underlying HMM state. To this end, hypotheses are stored as illustrated in Figure 144. That is, the original array contains now a lexicon state identifier but the actual score is now replaced by an index to a secondary array where active hypotheses are stored.

This secondary array contains the actual score and the secondary identifier (i.e. the LM context or the time start, depending on their use) and is sorted by this field. Since the number of active hypotheses is implicitly given by the index of the first hypothesis of the following active state, a sentinel value is required for the last active state.

The token recombination is performed by merging two sorted lists as illustrated in Algorithm 10. The use of sorted lists to store hypotheses and to take profit of this organization in a merge procedure is not novel and is described, for instance, in [Demuynck *et al.* 2000; Section 5.3] where both hash tables and sorted lists are proposed for their version of "per-state stack organization". The implementation of a lexicon decoder for a flat lexicon representation, using linear senone topologies as sub-word units and including the capability of managing several hypotheses could make use of the former merge operation. This operation is only required when two consecutive active states correspond also to consecutive states in the model since, in other cases, the simpler updating of the list (by taking into account either the loop or the outgoing HMM transition) is performed. The pruning has not been illustrated either in these scenarios. This algorithm would be quite similar to the actual one we have implemented, although we have directly made an implementation for the tree lexicon organization described below.

Jointly processing multiple frames

The possibility of jointly processing several consecutive frames is a novel idea that has been described in Section 10.6.9 (and illustrated in Figure 137). Although the main drawback of this idea is that their applicability is limited to those cases when the use of an external oversegmenter makes it possible to know in advance that no new hypotheses are incoming to the lexicon decoder root, we believe that several tasks, such as HTR (but, unfortunately, more limited for continuous ASR), may benefit from it.

Data: origin vector v, position of intervals $[x_1, y_1)$ and $[x_2, y_2)$
Result: destination vector v' is updated at interval $[x_3, y_3)$
$x_3 \leftarrow y_3 \leftarrow first\_available\_position$
$r_1 \leftarrow x_1$
$r_2 \leftarrow x_2$
while $r_1 < y_1$ *and* $r_1 < y_1$ **do**
  if $v[r_1].idx < v[r_2].idx$ **then**
    $v'[y_3] \leftarrow v[r_1]$
    $r_1 \leftarrow r_1 + 1$
  else if $v[r_1].idx > v[r_2].idx$ **then**
    $v'[y_3] \leftarrow v[r_2]$
    $r_2 \leftarrow r_2 + 1$
  else
    $v'[y_3].idx \leftarrow v[r_2].idx$
    $v'[y_3].scr \leftarrow max(v[r_1].scr, v[r_2].scr)$
    $r_1 \leftarrow r_1 + 1$
    $r_2 \leftarrow r_2 + 1$
  $y_3 \leftarrow y_3 + 1$
while $r_1 < y_1$ **do**
  $v'[y_3] \leftarrow v[r_1]$
  $r_1 \leftarrow r_1 + 1$
  $y_3 \leftarrow y_3 + 1$
while $r_2 < y_2$ **do**
  $v'[y_3] \leftarrow v[r_2]$
  $r_2 \leftarrow r_2 + 1$
  $y_3 \leftarrow y_3 + 1$

Algorithm 10: Merging sorted active hypotheses in an array.

As can be observed in Figure 137, we seek to directly compute the activation values at time $t + k$ from the activation values at time t, for a given value $k > 1$ (the case $k = 1$ being the usual one). To this end, we can assume that the likelihoods associated to HMM states at times $t + 1, t + 2, \ldots, t + k$ are available. Although the sparse array representing the active states of a linear senone sequence only contain the values at time t and $t + k$ (for the original and the destination arrays, respecively), the algorithm manages a vector containing at most k consecutive estimated values. When the following lexicon state is not active, its values are only computed from $t + 1$ to $t + k$ and so on.

A drawback of this technique (besides the obvious one of being restricted to some particular scenarios) is the fact that pruning strategies must consider states activated at different times. To this respect, we can use a sole threshold for all active states, which would lead to a too optimistic pruning for the more recent states, or to precompute a sequence of thresholds by taking upper bounds on the emission and transition probabilities used in the lexicon model. Obviously, the problem of using a too optimistic pruning is a matter of (both computational and spatial) efficiency and the generated states can also be pruned when using them in the next Viterbi step.

10.9 SPECIALIZED TREE LEXICON DECODERS

The lexicon decoders described in this section constitute one of the main contributions of this chapter and are used in the rest of the work. As previously explained, one of the main properties of linear lexicons is that, when the set of active states is contiguously stored sorted by the state identifier, the set of states reachable from them is generated in a sorted way. We would like to take profit of this property and other ones to extend the ideas from previous section to tree based lexicons. One of the first issues to consider is that trees can be serialized in several ways. Among them we can mention the well known depth-first and breadth-first (or level) traversals. The specialized Viterbi algorithms described here benefit from the fact that tree level traversals provide a topological order with the following features:

- the children of a given node occupy contiguous positions. The grandchildren also occupy contiguous positions, as can be observed in the example of Figure 145;
- if a subset of states is stored in this topological order and we generate the children of every active state following that order, the resulting list is also ordered. For instance, in the same example of Figure 145, if our set of active states is $\{1, 2, 4, 5\}$ and we generate their respective children, processing them in order, we obtain first $2, 3, 4$, then $5, 6$, followed by 8 and, finally, the empty set. This leads to the set $\{2, 3, 4, 5, 6, 8\}$ which is obviously ordered.

10.9.1 Model representation

Tree lexicon not tied to any particular sub-word repertoire

A tree model T of n states can be represented with several vectors of size n (one of them of size $n + 1$) as follows:

- **`from_prob`** stores the parent incoming transition probabilities. The root is an exception since it has no incoming edge;
- **`sub_word`** stores the index of the sub-word model incoming to this state. The root is also an exception in this case;
- **`first_child`** stores the index of the first child. The last child is deduced by looking the first child of the next state thanks to the topological sorting. This representation allows specifying an empty set of children. A sentinel in position $n + 1$ is needed for the last state. An example for the tree depicted in Figure 145 is shown in Table 5;
- **`first_out`** is used to specify the output words emitted by the tree model. This vector contains indices to a secondary vector storing words identities togetwher with their corresponding output probabilities. The size of this secondary vector, for the model of Figure 145 contains 8 positions. The reason to point to the first output word, as with the **`first_child`** vector (see Table 5), is the possibility for a single tree node to emit several words due to the presence of homophones in ASR or homographs in HTR, for example.

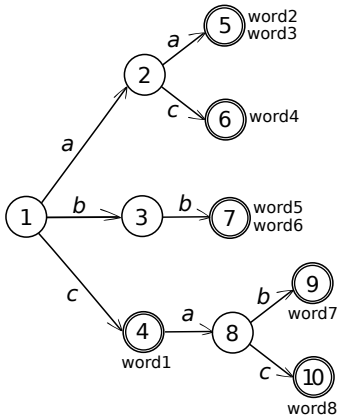


Figure 145: Example of a tree where sub-word units are just indices to the corresponding edges (or destination states that are univocally associated to them). The tree level traversal explains why children occupy consecutive positions. This representation is preferable over statically expanded models. Observe that when the successor states of an ordered subset are generated following this order, the resulting set is also ordered. Homophones and homographs explain the possibility of emitting several words at the same final state.

state	1	2	3	4	5	6	7	8	9	10	11
first_child	2	5	7	8	9	9	9	9	11	11	11
first_out	1	1	1	2	2	4	5	7	7	8	9

Table 5: Model representation for the tree example of Figure 145. Only the **first_child** and **first_out** vectors are shown. Observe the sentinel value at position 11.

We can guess that similar representations can be found in the literature. In particular, the idea of an index to the first transition has already been used in the LM representation described in Section 9.3.1.

unigram smearing

The vector of incoming probabilities is not strictly required since output words have attached probabilities and, moreover, unless working with unigrams, probabilities are tackled by the LM. Nevertheless, incoming probabilities can be used to implement unigram smearing (see Section 10.6.7) that can be removed at the outputs. The net effect is that unigram smearing probabilities dissapear, the tree model can be used without problems with an arbitrary LM while achieving a better pruning. Unfortunately, this approach cannot be extended *as is* to higher order LMLA.

Tree lexicon of linear senone sequences

When a lexicon tree is expanded with left-to-right acoustic HMM models without skips, the resulting model is a tree made of senones as depicted in Figure 146.

Although it is probably better to use a dynamic expansion of sub-word units for the sake of keeping a more compact model, it is useful to reason about trees made of senones units in order to extend the ideas of Section 10.8 that were described for left-to-right sub-word units. The following observations about the expanded tree models are straightforward. They are basically inherited from linear senone sequences but still holds when moving to trees:

- every state has at most two predecessors: itself and possibly his parent;
- if we ignore the loops, the expanded model is acyclic. Therefore, a topological order is possible in general.

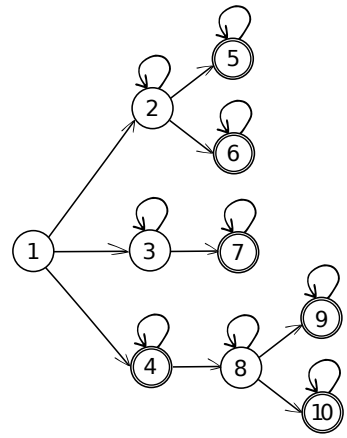


Figure 146: Example of a tree made of senones. This kind of trees is obtained by expanding a tree like that of Figure 145 with sub-word models made of senone sequences (Figure 92*e)*), although a dynamic expansion is preferable over a static one. As it can be observed, the information used to specify each state (excepting the root) include the score of the loop transition, the type of emission, the incoming probability and the index of the first child (the last one and number of children is implicitly given by the index of the first child of the next state, hence the use of a sentinel value at the end).

This allows us to propose an algorithm similar to that of Figure 140 where loops and children are considered separately to obtain sorted lists to be merged afterwards or, better, try to obtain a more efficient specialized algorithm.

The model representation for a statically expanded model is essentially the same as in a general tree model although, instead of using the **`sub_word`** vector to store the index of the sub-word model, **`e_index`** stores the index of the associated emission probability class associated to the emitted frame (e.g. acoustic frame, in the case of ASR) to be observed. Besides the use of the vector **`from_prob`** to store incoming transition probabilities,[60] another vector called **`loop_prob`** is used to store the loop transition probabilities.

Although statically expanded models have the advantage of making transparent the use of sub-word units with a different number of states and the use of within-words context-dependent units,[61] the advantage of the dynamic expansion is twofold: on the one side, the representation is more compact. On the other side, it is independent of the chosen sub-word unit models. Indeed, we have currently used memory mapping[62] to represent these trees that may become quite large for huge vocabularies. Another problem of the static expansion is that linear sequences inside sub-word models are represented in the same way as transitions from a sub-word model to its children, so that there are many states with only one child. To this respect, we can mention a slightly different representation found in the literature [Novak *et al.* 2003]: *"The cost of traversing the tree can be minimized by factoring the tree into a set of linear state sequences"*. We can understand that some sequences of tree nodes with just one children deserve a special representation similar to a linear lexicon and this would even improve data locality. Nevertheless, we can achieve similar and additional advantages when using the dynamic approach.[63]

memory mapping

60 Incoming probabilities are required even when unigram smearing is not used.

61 The case of across-words context dependent units is discussed elsewhere (in this chapter and in other parts of this work).

62 Memory mapping was also used with the LM, as described in Section 9.3.2.

63 We have not investigated the relevance of the presence of linear sequences in the tree lexicon made of sub-word unit references (i.e. without expansion).

10.9.2 Representation of active states

Some prior art

We must distinguish the representation of the model and the set of active states. Since tree lexicons are one of the most widespread used lexicon schemes, it is reasonable to ask ourselves how are active states represented in alternative systems.[64] To be honest, these implementation details are mostly skipped in the literature, paying more attention to the model representation and talking in a general way about token passing. Although it is possible, in principle, to download some open-source recognition engines in order to inspect the code to obtain more details, we have not performed this error prone task in general.[65]

attached to the model

We have also to take into account the fact that many recognition engines have a different organization than ours. For instance, many systems use the LM-conditioned approach based on tree-copies. In this case, the representation of active states is tailored to the overall system rather than to a generic lexicon decoder as in our case. For these systems, it is usual to attach tokens to the tree model to represent active states, as is the case of [Demuynck *et al.* 2000; Section 5.3] where, in its "sorted lists" version the tokens are organized as sorted lists attached to the nodes of the lexicon network. A similar idea is described in [Huijbregts 2008] although tokens are sorted by their score. In both cases, several tokens can be attached to the same tree node since they belong to different tree copies. The main idea is, in any case, to use a dense array of size the number of tree nodes instead of resorting, for instance, to hashing. This array is unique for the entire recognizer and, in some cases, makes part of the tree model representation.

hierarchy of data structures

A different representation is described in [Ortmanns 1998] for a tree-copy[66] approach to LM-conditioned decoding: different types of hypotheses (trees, tree states, arcs and HMM states) are maintained in different data structures constituting a hierarchy where each hypotheses from one level is associated to a contiguous interval of hypotheses of the next one, as described in Figure 147. Two different arrays of each type are used as is usual in ephemeral or in-place updating implementations.[67] Two data traversal steps are performed at each Viterbi step as described in [Ortmanns 1998; Sect. 3.2] and in [Sixtus 2003; Sect. 7.7], and illustrated in Figure 136.

Let us finish our brief review on active state representations by discussing an approach more related to our proposal. The active envelope algorithm [Nguyen *et al.* 2000; Nguyen 2002] is a specialization of decoding to the case of a tree lexicon with left-to-right HMM units. The algorithm basically stores the active nodes of the lexicon prefix tree in a linked list following a reverse breadth-first traversal from the deepest levels to the root. This ensures that parents are traversed in such

64 Other relevant brand of decoding systems rely on transducer compositions.

65 An exception has been the iAtros recognizer [Luján-Mares *et al.* 2008], mentioned before in this chapter, which uses a heap to store the states from a recognition stage and a hash table to locate them. The approach of studying source code cannot be applied, obviously, to closed source recognition systems.

66 It has been extended to general WFSTs in [Rybach 2014; Section 6.8] although the basic idea remains. We have also consulted [Schlüter and Ney 2010].

67 That is, interleaved for t and $t+1$ as illustrated in Figure 128b.

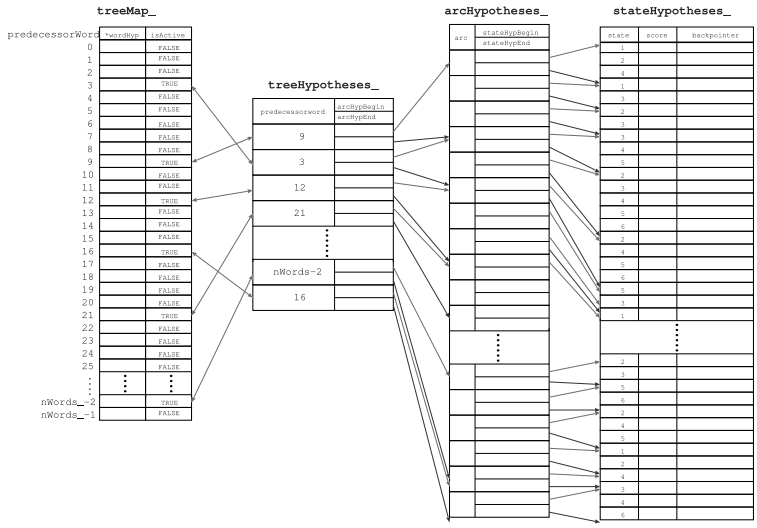


Figure 147: Figure from [Schlüter and Ney 2010; Slide 218] (essentially similar to [Ortmanns 1998; Fig. 3.7] and [Sixtus 2003; Fig. 7.8]) illustrating the representation of tree, arc and HMM active state hypotheses in RWTH decoder.

an order that, when they generate their children, the same order of the original data structure is produced. A Viterbi step is performed by traversing this list updating hypotheses, including new ones and removing others due to the use of pruning techniques. This step requires to traverse two consecutive tree levels simultaneously with the use of two pointers. This allows to perform a merge of two ordered lists: the updated children generated by the parents and the previous children. In this way, each node is traversed twice (one from each list). This algorithm can be seen as a specialized implementation of the reversal topological traversal or "piggy-backed" technique described in Section 10.3.3. The main similitude with our algorithms is the fact of maintaining active states sorted by a level traversal of the lexicon tree. Let us remark that we have been aware of this algorithm only when our decoders were already implemented. Despite this resemblance, there are also some differences: on the one side, it is based on linked lists instead of using arrays consecutive in memory. On the other side, it performs a destructive updating instead of the persistent approach where a new version of the set of active states is obtained while maintaining the original one.

Proposed active state representation

flatter representation

The proposed active state representation is essentially the same as in the case of linear lexicons: a sole array of active hypotheses contiguously stored and sorted by state identifier. Our representation for active states is flatter than, for instance, that of [Ortmanns 1998; Sect. 3.2] (Figure 147) since only a list of active HMM states is maintained sorted by state identifier. This array is constructed in a single step instead of executing a secondary traversal to compact hypotheses due to pruning. Instead, two different pruning criteria are used simultaneously:

1) when creating an active state to be stored in the array, and 2) when this list is used as the origin in the next Viterbi step.

The basic array representation mentioned so far is suitable for a statically expanded tree model where sub-word units are linear senone sequences. In that case, each array cell would contain: the state identifier, a score and, in most cases, a back-pointer identifier. Nevertheless, some variants to this representation (similar to those described in Section 10.8) can also be envisaged:

- the common case where we are jointly managing multiple tree lexicon instances in a sole decoder would adopt the scheme of Figure 144 where each active state may contain several hypotheses distinguished by their LM context or by their time-start depending on the actual use (for LM-conditioned and for time-conditioned decoders, respectively). In that case, a double ordering allows us to apply the idea of merging sorted lists at both levels, as described before;
- a flatter representation of the previous idea can be used so that a sole memory block is used to store what in Figure 144 constitute two different ones. As mentioned in Section 10.8, this approach is compatible with the idea of using different sub-word types even with different implementations provided that a common interface allows the proper serialization/marshaling of their active state representation;
- it is also possible to manage the entire set of active states of a sub-word unit in a joint way. This would make it easier the use of complex sub-word topologies as previously explained for the case of linear lexicons. This sub-word-centric approach is described in Section 10.9.8.

10.9.3 Different traversal possibilities

We can consider two different *basic* strategies with pros and cons:

FORWARD TRAVERSAL implies going from the root towards the leaves;

BACKWARDS TRAVERSAL means processing first the active states closer to the leaves until reaching those near the root of the tree.

Centering our attention on the case of trees made of senones, the main idea is that states are traversed and each active state is used:

- to produce its updated version;
- to generate its children.

Note that each of these possibilities generates states in a sorted way. Since the same state might have been alive and be activated by its parent, these different states have to be merged in order to obtain an ordered list from the previous ones and to combine the tokens (scores and back-pointers) of states simultaneously activated by both cases. The final result is a sorted list of active states contiguously stored in an array. This procedure, crystal clear when using trees made of senones, is slightly more complex with sub-words of a more complex topology.

The basic idea for dealing with complex sub-words units is related to a way to implement the dynamic expansion. Indeed, although it is possible to implement the dynamic expansion for trees made of senone sequences in another way, the following approach is, in our opinion, quite reasonable and has been partially introduced in Section 10.8. It consists in storing and processed together the active states of each instance of sub-word unit. Indeed, we can consider the active states representation of a sub-word unit as an entity (e.g. a C++ class in our `April` toolkit) with the following features:

- it is possible to obtain an updated version that is placed (in a serialized way) in the destination array of sorted consecutive states;
- to this end, we only need the emission probabilities and the tokens incoming to the initial state;
- the representation of a new sub-word unit instance can be constructed and, in that case, only the initial set of incoming tokens is required;
- it is obviously necessary to extract the final tokens outgoing these models in order to feed with them other sub-word instances.

We will discuss this approach later in the corresponding forward and backwards versions of the algorithm, but let us first briefly anticipate some advantages of forward and backwards traversals as well as presenting two other possibilities.

backwards traversal facilitates parallelism

One of the main advantages of the backwards traversal is that the tree root is the last processed state. This means that, in a re-entrant lexicon decoder with token passing (as depicted in Figure 125), the incoming token required by the tree lexicon decoder is not needed *until the end*. This makes it possible to parallelize the overall decoder by executing the lexicon decoder in a thread and the LM updating procedure in another one, as described in Section 12.3.

forward traversal enables error correction

On the other side, the forward traversal seems the only reasonable way to include null transitions and error correction edition operations since the hypotheses being generated are used during the algorithm. This has been briefly addressed in Section 10.8 and will be discussed below for the case of trees.

The forward traversal can also be used in parallel with the LM updating procedure, when non using null transitions or error correction, at the expense of a more cumbersome implementation. The problem is that hypotheses incoming to the tree root are not known at the beginning. However, it suffices to delay the evaluation of the root's children to the end. This kind of patch is less elegant and more cumbersome.[68]

combining forward & backwards

A third alternative consists in combining forward and backwards traversal applied on the same tree lexicon network until both traversals meet at some central point. The advantage of this procedure is that two different threads can proceed in parallel, basically doubling the speed. This possibility is described in Section 10.9.7. In any case, we have to point out that there exist another way to parallelize lexicon decoders: by splitting the trees at the root (this is described in Chapter 12 at

68 We cannot foresee any advantage of this approach other than combining it with backwards traversal as mentioned below.

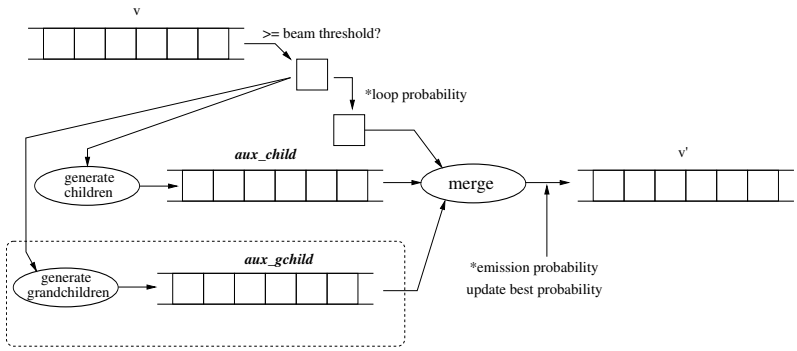


Figure 148: Forward traversal Viterbi algorithm for tree lexicon made of senones and using sorted arrays and an auxiliary queue. A secondary auxiliary vector may be required for Bakis topologies.

Section 12.3). Fortunately, both approaches are compatible and can be combined.

sub-word-centric approach

We can also mention that there is a fourth alternative that we have called "sub-word-centric" which consists in maintaining a sole decoder instance for each sub-word type. This procedure, described into more detail in Section 10.9.8, is slightly more complex since we need a set of active state arrays/queues, one for each different type of subword unit. The hypotheses outgoing each sub-word model are accumulated in the queue of the corresponding children. The idea of taking profit of the topological order remains only *partially*: on the one side, hypotheses of each sub-word unit are generated in a sorted way. On the other side, the recombination of tokens coming from different subword units cannot be merged in a so trivial way since they are only sorted when associated to the same tree lexicon node but not to the same sub-word unit.

Let us see how these different traversal procedures can be implemented and, for the particular case of forward traversal, let us devote a dedicated section to explain how to deal with error correction edition operations.

10.9.4 Forward (root to leaves) traversal

Forward traversing a senone tree using an auxiliary array

Let us first see a forward traversal algorithm for dealing with trees made of senones. An active state or token is basically composed by an index state and a score (i, s) (back-pointers are considered as dummy values that are simply copied when combining tokens). Active states associated to time t are stores in array v and the purpose of a Viterbi step consists in creating another array v' associated to time $t+1$ given v, the tree model and the vector of emission probabilities. An auxiliary queue aux is used to store temporally the scores of states produced by the transitions from parent to child.

The basic idea of this algorithm, illustrated in Figure 148, consists in processing the states from v in an ordered way (in the direction

```
Data: origin vector v, destination vector v', incoming tokens itk
Result: destination vector v' is updated
aux ← an empty queue
if itk is not empty then
    place itk as if it was the content of v at root's position
for st ∈ v do
    if st is not pruned then
        generate children of st and place them in aux /* for loop
        multiplying score by incoming trans probability */
        update st with loop transition probability
        while aux not empty and its front is strictly below st do
            extract front element from aux and place it at v'
            /* emission is applied before storing result */
        if aux not empty and its front is equal to st then
            extract front element from aux, merge it with st and
            place it at v'
            /* emission is applied before storing result */
        else
            place st at v'
            /* emission is applied before storing result */
while aux is not empty do
    extract front element from aux and place it at v'
    /* emission is applied before storing result */
```

Algorithm 11: Viterbi step using a forward tree traversal on a tree lexicon made of senones and using an auxiliary queue.

from the root towards the leaves), applying the pruning to these states. Each time a state st is processed, their children are generated and are inserted into aux queue. After that, we proceed to extract from aux all active states whose identifier is strictly lower than st in order to place them into v′. When these states have been processed, the next element in aux to be processed, in case it is not empty, is greater than or equal st. In case it is equal, both token are combined[69] and placed into v′. A pseudo-code is shown in Algorithm 11.

Tokens incoming to the tree model from the outside (e.g. in a reentrant network scenario as depicted in Figure 125) are used as if they were placed in the vector v associated to the root node identifier: the are used to generate hypotheses associated to the first phonemes/graphemes of each lexicon word.

Whenever an active state is placed in v′, the emission probability associated to it is applied, and the best probability is updated. This value is used to obtain the beam threshold for the following Viterbi step. It is easy to extend this approach to more complex pruning criteria (e.g. histogram pruning).

This algorithm, as with all reasonable implementations of Viterbi algorithm, is linear with the number of active states and the question

69 In this simpler case, the one with best score is preserved.

Number of states	Hash swapping	Active Envelope	Forward traversal
9,571	3.001	16.082	29.350
76,189	2.761	12.924	28.036
310,888	1.922	6.442	24.534

Table 6: Experimental results of Viterbi algorithm based on forward traversal of a topologically sorted tree compared with the use of a hash table and with the Active Envelope algorithm. Results shown in millions of active states or hypotheses updated per second (the higher, the better). Table taken from [España-Boquera *et al.* 2007; Table 1].

that remains is the relative constants involved. A better cache performance and an internal loop with less overhead is obtained compared to other algorithms that use linked lists or use hash tables to store and look up the set of active states. Therefore, the asymptotic cost is the same but a practical speed-up is expected. In order to obtain some empirical insight, we have compared this algorithm with a conventional Viterbi algorithm based on hash tables with chaining to store and to look up the active states and the active envelope algorithm described in [Nguyen *et al.* 2000]. All algorithms have been implemented in C++ and use the same data structures to represent the tree models.

experimental results

These experiments, published in [España-Boquera *et al.* 2007], were done on a Pentium D machine at 3GHz with 2 Gbytes of RAM using a Linux with kernel `2.6.18` and the `gcc` compiler version `4.1.2` with `-O3` optimization. The size of these trees varies from 9,571 to 310,888 states. These lexical trees were obtained by expanding 3-state left-to-right without skips which, as described before, are commonly used in ASR tasks. These acoustic models are hybrid neural/HMM models. Only the Viterbi decoding time has been measured (the emission scores calculation and other preprocessing steps were not taken into account). The result, shown in Table 6, demonstrates the validity of the proposed algorithm. Nevertheless, this algorithm has been superseded by other designs described below.

Forward traversing a senone tree using two pointers

One of the obvious improvements over the above algorithm consists in removing the auxiliary queue. This can be done by using two different pointers (e.g. numerical values indexing the array) as follows:

- one of the indices points to the hypothesis whose children are being generated;
- the other is used to locate the element to be merged with each generated child.

An scheme of this variation is illustrated in Figure 149 together with an informal explanation while a pseudo-code is shown in Algorithm 12.

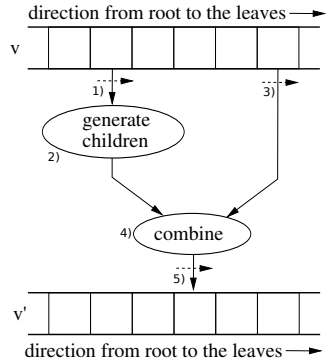


Figure 149: Forward traversal Viterbi algorithm for tree lexicon made of senones and using two pointers: Index (1) points to the element of v vector whose children are being generated at (2). Index (3) seeks the corresponding child. The main loop traverses 1) and a nested loop advances (3) updating elements (placing them at (5)) while they are lower than the expected child. If the child is found, it is combined (4) with the generated child, otherwise ((3) exceeds the child value) the child is placed at (5). (1) advances when all their children have been generated.

Data: origin vector v, destination vector v', incoming tokens itk
Result: destination vector v' is updated
i_child $\leftarrow 0$
if itk *is not empty* **then**
  place itk as if it was the content of v at root's position
for (i_parnt $\leftarrow 0$; i_parnt $< v$.size; i_parnt $\leftarrow$ i_parnt $+ 1$) **do**
  st $\leftarrow v$[i_parnt]
  for child $\in$ *children of* st **do**
    while i_child $< v$.size $\wedge$ (x $\leftarrow v$[i_child], x.st $<$ child) **do**
      update x with loop probability and place it at v'
      i_child $\leftarrow$ i_child $+ 1$
    if i_child $< v$.size $\wedge$ (x $\leftarrow v$[i_child], x.st $==$ child) **then**
      update x with loop probability
      combine x with child and place it at v'
      i_child $\leftarrow$ i_child $+ 1$
    else
      place child at v'

Algorithm 12: Viterbi step using a forward tree traversal on a tree lexicon made of senones using two pointers.

Extension to more complex sub-word units

The former algorithms can be trivially adapted to perform a dynamic expansion of the trees made of senones using the same techniques described in Section 10.8 for linear lexicons. There exist at least two different possibilities:

- the model is dynamically expanded but their structure is the same as if the process was statically performed. In that case, the algorithm would remain essentially the same and only the use of the model would change;
- it is also possible to make a version of the forward traversal algorithm where the active states of each sub-word unit are processed together. This version seems more suited to extend the algorithms to more complex sub-word units.

Now, let us see into more detail the last option. As explained in Section 10.9.3, the active states associated to a sub-word unit expanded in a lexicon model can be considered as an object with methods to obtain an updated version (that is conveniently serialized and stored in a destination vector v'), to construct a new instance or to consult the tokens outgoing the model required to create or update their successors.

The previous algorithms based on the use of an auxiliary queue (Algorithm 11) can be adapted to this scenario: each time an state st is processed from v, tokens outgoing st are extracted and adapted to become the input associated to each child by including the corresponding transition probabilities. These are the tokens placed in the aux vector associated to each child. After processing the children of st, the tokens from aux vector are processed by while extracting values strictly lower than st in order to activate new sub-word units. When finishing this step, the sub-word representation associated to st is updated and placed at v'. In order to perform this updating procedure, we need tokens incoming to the model. These tokens are extracted from aux, when present, or are empty in other case. As with the original algorithm, the tokens incoming by the root to the overall tree lexicon (from the re-entrant model) are used as if the root node was an active state that, indeed, is not associated to any sub-word unit.

As can be observed, only the lightweight tokens outgoing each sub-word model are written in aux while the more heavyweight model representation is kept in v and processed first to extract the outgoing tokens and later to write its updated version at v'. The advantage of this version is clear even when using linear senone sub-word topologies since, in that case, the entire set of active states can be processed in a more efficient way using the algorithms described in Section 10.8.

Relating the version of forward traversal based on two pointers, formerly described in Algorithm 12, the model pointed by `i_parnt` in vector v is used to extract the outgoing tokens while the model pointed by `i_child` is updated using the incoming tokens extracted from the parent and adapted to this child. Only when this pointer reaches the end of the array or the pointed element surpasses the corresponding child, a new instance for the active state representation of the child is created using the same incoming tokens as in the previous case.

How to extend the above algorithms for Bakis topologies

It is possible to extend the above algorithms to Bakis HMMs. The model representation of a tree lexicon statically expanded with Bakis sub-word units is similar to the case of linear senone sequences excepting the inclusion of a new vector **`skip_prob`** which stores, for every state, the incoming skip transition probabilities. In order to iterate over the set of grandchildren of a given state **`i`**, the algorithm loops from **`first_child[first_child[i]]`** to **`first_child[first_child[i+1]]-1`**.

The Viterbi algorithm based on a forward traversal of such a model, when using an auxiliary queue, is similar to the previous case although another auxiliary queue (depicted in Figure 148, where it is called **`aux_gchild`**) is now used to store the information associated to the grandchildren. Now, the merge procedure has to take both queues into account.

The extension to the dynamic expansion of sub-word units, using an auxiliary queue, could be done in the same way[70] if this kind of transitions appeared before expanding the model.[71] However, the expression "Bakis topologies" refers to the sub-word models (we are still expanding an ordinary tree model). This is the reason why the proposed solution for the dynamic expansion differs from the previous one and consists in considering two different entry points and two different output points at each sub-word model object instance, just as described for the case of the expansion of linear topologies of Section 10.8 in Figure 143.

The versions based on two pointers for trees statically expanded with Bakis models can be extended to three pointers, although this is more cumbersome than expected: the index pointing to grandchildren is the one that we are moving while creating elements at the destination vector v'. This process is guided by the other pointers that are jointly managed in order to determine the element we are seeking in the aforementioned pointer. We rather recommend go for the dynamic expansion using two pointers: We can make here the same observation regarding the fact that Bakis topologies are considered at the sub-word models and not at the tree model. The algorithm remains essentially the same as the general dynamic expansion with two pointers but not sub-word instance objects would have to deal with two different entry/exit points.

10.9.5 Decoding sub-word DAGs with error correction

The forward traversal Viterbi algorithm can be extended to deal with null transitions and with error correction edition operations as those described in Section 10.4. The basic approach is similar to the case of linear lexicons (see Section 10.8).

Error correction edition operations make more sense at the phone or grapheme level than at the frame level. This is the reason why, instead of processing frames, we will consider a lexicon decoder capable of processing a DAG made of sub-word units. These lexicon decoders could be used in the generation of a word DAG from a sub-word DAG (e.g. a grapheme DAG in HTR or a phone-graph in ASR). The DAG generation algorithm to obtain the word DAG is described in Section 11.2, but the lexicon decoder described here constitute a useful piece for that algorithm.

As described in Section 10.2.2, we have to provide two different procedures that must be applied following the input DAG topological order: the **`vertex-step`** procedure must be applied to every vertex before using this set of active states as the origin of an edge **`edge-step`** procedure. Put in other words, we are not seeking at processing an entire sub-word DAG at a time but, rather, to provide the basic pieces that are applied at each edge and vertex associated to the input DAG, as illustrated in Figure 123. Let us see into more detail these operations.

70 We only have to take into account that the merge procedure is applied to both auxiliary queues on the fly (combining tokens when they appear simultaneously) in order to traverse them as if it was just one queue.

71 The model would no longer be a tree but a very particular case of lexicon network. Specialized decoding algorithms for lexicon networks are the subject of Section 10.10.

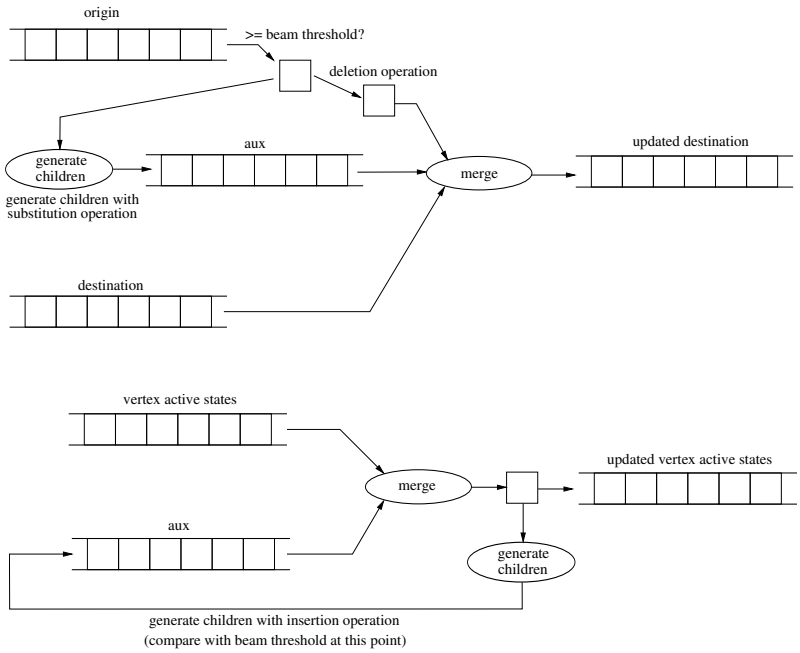


Figure 150: Two basic procedures for using Viterbi to process an input DAG with error correction using an auxiliary queue: **`edge-step`** (*top*) and **`vertex-step`** (*bottom*).

EDGE-STEP For every edge, a Viterbi step takes the set of active states of the origin vertex and use them to update the active states of the destination vertex. This procedure only considers the cost of deletions and substitutions. As can be observed in Figure 150 (top), this algorithm is similar to the forward-traversal algorithm of Section 10.9.4 where loop probability updating is replaced by the deletion operation, whereas the generation of children states corresponds to the substitution operation (including a symbol by itself or a correct transition). Another difference, which can be also used in previous algorithms to process a DAG as input data, is the presence of second input queue which stores the active states already present at destination vertex. These active states should have probably created by means of other edges incoming to the same destination vertex. These values are simply merged and this queue is not needed, obviously, when the input data is a sequence. The version using two pointers is similar to the scheme depicted in Figure 149 although a second input vector containing the active states already present in the destination is provided (Figure 151 *(left)*). A third pointer is used to traverse the new input vector and it is advanced on demand to merge tokens each time a new state is to be included in the "updated destination" vector.

deletions & substitutions

VERTEX-STEP Once all edges arriving at a given vertex have been processed, the insertion operation is considered. This operation updates a set of active states without consuming any symbol. As can be observed in Figure 150 (bottom), the output of the auxiliary queue is

insertions

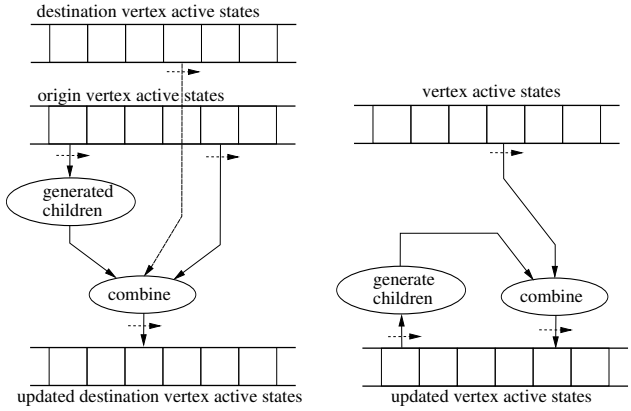
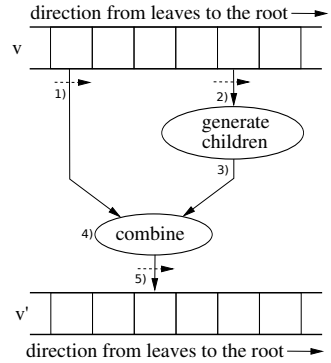


Figure 151: Two basic procedures for using Viterbi to process an input DAG with error correction with pointers (without auxiliary queues): **`edge-step`** (*left*) and **`vertex-step`** (*right*).

Figure 152: Backwards traversal Viterbi algorithm for tree lexicon made of senones and using two pointers: Index (1) points to the element being updated and, perhaps, combined with the children generated at (3). Index (2) drives the main algorithm and is only used to generate the children of states active in v. When one of these children is not found by (1), a new state is created (4) and (5). As can be observed, the last processed element of v is the one nearest to the root.



used to insert more active states in the same queue to take into account the possibility of several insertion operations. The case without auxiliary queue is also very simple: two pointers are required but, now, one of them traverses the set of states that is being created (Figure 151 *(right)*). Relating the cost of this procedure, although several states can be activated with consecutive insertions from a given active state, the number of generated successor states is obviously bounded by the total number of states.

10.9.6 Backwards (leaves towards the root) traversal

Backwards traversal means processing the lists of active states in a reversed way, meaning that children are visited before their parents. Put in other words, we start from leaves towards the root.

Our aim is to adapt the version of our previous forward traversal algorithm using two pointers. As with the forward traversal version, each element is updated in the destination array when it is considered to be a child, while the parent pointer is only used to determine which children are to be activated form it in order to provide an incoming token either to combine with the existing child or to create

```
Data: origin vector v, destination vector v', incoming tokens itk
Result: destination vector v' is updated
i_child ← 0 /* points to the farthest from the root */
if itk is not empty then
    place itk as if it was the content of v at root's position
/* from the farthest to the closest to the root */
for (i_parnt ← 0; i_parnt < v.size; i_parnt ← i_parnt + 1) do
    st ← v[i_parnt]
    for child ∈ children of st do
        while i_child < v.size ∧ (x ← v[i_child], x.st > child) do
            update x with loop probability and place it at v'
            i_child ← i_child + 1
        if i_child < v.size ∧ (x ← v[i_child], x.st == child) then
            update x with loop probability
            combine x with child and place it at v'
            i_child ← i_child + 1
        else
            place child at v'
```

Algorithm 13: Viterbi algorithm for tree lexicons based on a backwards tree traversal, taking profit of the topological order and using two pointers. Remark the similarities with Algorithm 12.

it, as illustrated in Figure 152. An informal explanation is shown in Algorithm 13.

This particular version has more resemblances with the Active Envelope from [Nguyen *et al.* 2000] than our previous forward traversal, although it remains different due to the use of a different set of active states since, for instance, it is not an in-place updating algorithm.

The main advantage of this procedure is that the input tokens incoming to the lexicon network are only required at the end. This allows a simpler parallelization of the lexicon network decoding and the LM updating using threads, as described in Section 12.3.

This algorithm can be extended to Bakis topologies and to use a double ordering of states to jointly manage multiple hypotheses in a way similar to the previous forward traversal algorithm.

10.9.7 Combining Forward and Backwards

The basic idea of this algorithm consists in combining the previous forward and backwards traversals applied on the same array representing a set of active states. The main reason to do that is probably to allow different threads to proceed in parallel since, in other case, any of the above algorithms are simpler. The reasonable choice, for both traversal directions, is to use the algorithm based on using two different pointers traversing the origin array.

In order to avoid the overhead caused by mutexes (or similar things to synchronize the associated threads) the set of active states can be divided beforehand by taking into account the different relative speeds

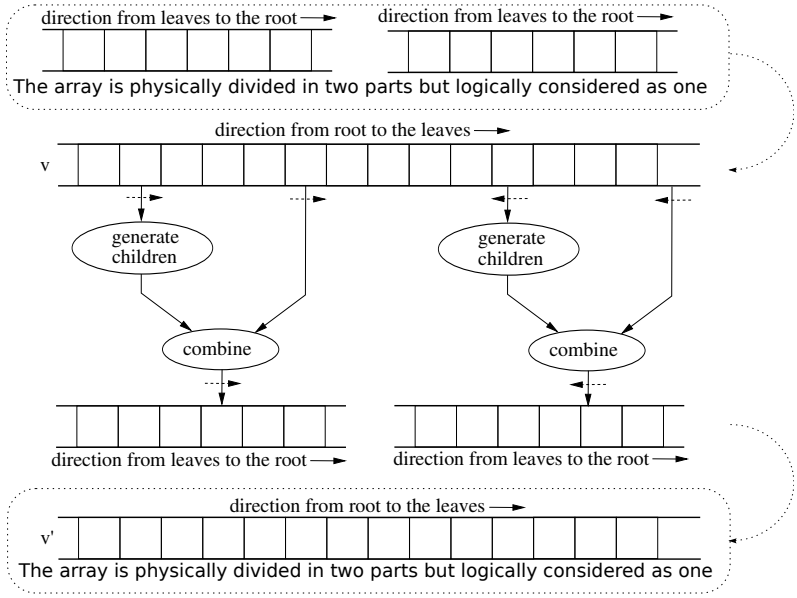


Figure 153: Combining Forward and Backwards traversals.

of each version of the algorithm. The only synchronization penalty would be at the end.

The resulting destination array/queue of active states is naturally divided into two arrays since we cannot precompute the exact sizes generated by each part. Therefore, we have to assume that the input array is also split since it is the result of a previous Viterbi step, as illustrated in Figure 153. This makes the implementation a more cumbersome and may have a slight overhead.

It seems convenient to remark that there is another commonly applied technique to parallelize the decoding of tree lexicon networks that seems perhaps easier: it consists in breaking the tree intro disjoint branches (breaking at the root level). However, we cannot discard the novel proposed technique and both techniques could be combined.

10.9.8 Sub-word-centric decoding

The basic idea of sub-word-centric decoding is to jointly process the active states associated to each different type of sub-word unit. In the case of context dependent units, we would jointly process the models associated to each possible context dependent unit since we want to take profit of the fact that all the hypotheses processed together make use of the same model (topology and output emission). Each of these units would have its own array of active states sorted by the lexicon state identifier,[72] as illustrated in Figure 154.

Although active states within each model are sorted and the set of children generated by each model is also generated in a sorted way, the

72 We could also maintain a double ordering arrangement when jointly managing multiple hypotheses from several decoding instances: First sorted by lexicon state identifier and later by the LM identifier or by the time-start, depending on the type of recognition engine.

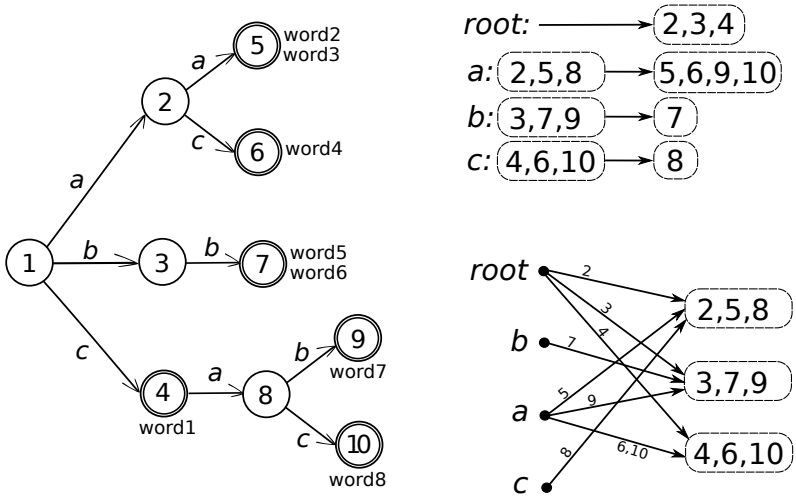


Figure 154: Explanation example of sub-word-centric decoding: on the left, the tree example of Figure 145, on the top right we can observe the set of lexicon states associated to each sub-word model instance and the sorted list of generated children produced by each model, and, finally, on the bottom right, the n-way merge produced by a bucket-sort inspired algorithm that processes the outgoing hypotheses in a pre-determined way and generates and stores the set of children in a finite set of destination queues. In this particular case, the order root, b, a, c allows us to use just one queue for each sub-word model instance although, in general, more than one queue might be required for a given sub-word model.

problem is that these children belong, in general, to states that have to be processed by different models.

The proposed decoding technique is obviously based on the dynamic case, meaning that the tree model is not statically composed with their corresponding sub-word models. Among the possible advantages of a sub-word-centric approach, we can mention:

- it is reasonable to use specialized decoders for each sub-word model topology. Moreover, loops can be expanded at compile time and transition probabilities can be hardwired. Since a sole model is used to process multiple instances, the scheme commonly used for jointly managing multiple hypotheses is the reasonable choice and the overhead of using a dense representation of states[73] is negligible;
- the management of output emission probabilities is also more centralized;

The main expected drawback of this approach is the management of tokens outgoing a sub-word model that have to be redirected to the corresponding incoming one. We have to remark that these tokens usually correspond to a small proportion of the overall hypotheses, most of them being inside the sub-word model instances.

The first naive solution that comes to our mind to solve this redistribution of tokens from one lexicon tree node to its children is to

73 An explicit list of hypotheses associated to each sub-word model state even is this list is empty in some circumstances.

construct n^2 different queues, being n the number of different sub-word model types. These queues would receive the hypotheses from a particular sender to a known receiver. Each sub-word model instance would have to perform an n-way merge of all hypotheses sent to it. This process can be achieved by using a selection tree or a min-heap [Knuth 1998; Sect. 5.4.1]. A destination array is not required since the data would be processed on the fly. However, since the set of elements to be merged is finite and the sets are disjoint, we propose the following alternative solution with some resemblances to bucket-sort:

- when a sub-word model has been processed, the outgoing hypotheses are also generated. They are used for two different purposes:
  1. to compute the output hypotheses of the lexicon tree associated to final states; and
  2. they are required for the next Viterbi step iteration, and will be sorted in an array.[74] Note that they are generated in a sorted way associated to the state indices of the internal sub-word model.
- at the beginning of the next Viterbi step iteration, we have to process these arrays in order to generate the hypotheses incoming to the corresponding children. We will assume that these arrays are processed in a particular order, which allows us to pre-compute a finite set of queues and to assign each one to a particular child. Put in other words, when the set of children of an active state is generated, the destination queue of each child is a data available in the model;
- note that the number of arrays to implement the queues might be much lower than the actual number of queues, since we can also determine when a queue will no longer be written;
- instead of performing an n-way merge, the input tokens are taken from the corresponding queues in the pre-computed particular order.

The pre-computation of the order in which sub-word models are processed to generate the children hypotheses as well as the assignment of these children to their corresponding queues can be easily done by using a greedy approach where the set of destination children is processed sorted by their lexicon state identifier. The lowest state to be processed first determines the order in which sub-word models are processed, while new queues are created on demand when there may be an intermediate belonging to this sub-word model that is not yet written.

Although the greedy algorithm considers that all states are active, which is quite improbable in practice, the resulting order and queue assignment still works when applying it to a subset of active states, as can be easily deduced. The risk is that, perhaps, some of these queues are empty and this, although correct, might produce a negligible overhead since the number of queues is expected to be quite low.

74 In many cases, this array already exists since it is the set of hypotheses associated to the final state of the sub-word model excepting an offset that can be taken into account when computing the children hypotheses.

10.10 LEXICON NETWORK (DAGWS) SPECIALIZED DECODERS

Lexicon networks are acyclic models which cover linear and tree lexicons as special cases. As with previous models, they are only acyclic when expressed as a network of sub-word units, but not necessary when those models are expanded.[75]

Observe that general decoding techniques, as those described in Section 10.3, can be used here in an straightforward way. Moreover, the use of active states representation seems mandatory because these networks can be quite huge and only some states are active in practice. Therefore, we could basically resort to those techniques where a list of active states is maintained and either an array of size the number of states or other more compact dictionary techniques (e.g. hashing) are used to determine if an state is already active and, in this case, to locate it.

Since those general techniques are already known by the reader and it makes non sense to repeat them here, the interest of this section is to briefly discuss other techniques. We propose to briefly address how the techniques proposed for tree lexicons could be adapted to more general DAGWs in spite of the fact that the basic properties that make them possible does no longer hold.

10.10.1 Adapting forward traversal from trees to DAGWs

Although DAWGs are still acyclic, several properties used in previous tree lexicon decoding algorithms no longer hold:

1. the identity of a word cannot be always determined by the final reached state, which means that we have to mark in some way[76] the tokens traversing the network in order to determine the output word identity;
2. some states can have more than one parent; and
3. there is no sorting of states ensuring that the children are also generated in the same order which, a priory, invalidates the tree level traversal algorithms of previous section.

Despite this non-encouraging scenario, let us try to adapt tree-lexicon decoders to deal with DAWGs. A first step to relate general DAGWs with trees consists in applying a depth-first search (DFS) of the lexicon network. DFS can be used to classify the edges of a directed graph, as explained in [Cormen *et al.* 2009] and illustrated in Figure 155, into:

TREE edges, which are those used for exploring the graph,
FORWARD edges, which do not belong to the tree and point to a descendant other than a child,
BACK edges should never appear in an acyclic network
CROSS edges point to a neither an ancestor nor a descendant.

75 Static expansion is not necessarily the case, dynamic expansion is usually the most reasonable choice.

76 The algorithms proposed in this section only need to track the traversal of forward and cross edges to determine, together with the final state, the word identity.

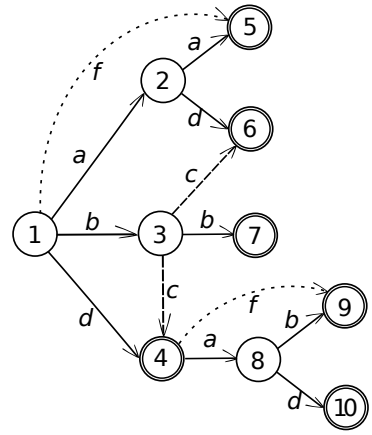


Figure 155: Example of a lexicon network. States have been numbered according to a tree level traversal of the *depth-first tree* embedded in the network. Lexicon network edges are depicted depending on their type after a DFS classification: solid lines correspond to tree edges, dotted lines to forward edges whereas dashed lines correspond to cross edges. The fact that lexicon networks are acyclic explains the lack of back edges, whereas the application of a first topological sort explains why DFS has produced cross edges oriented from lower to higher vertices.

The depth-first forest of the lexicon-network is the part composed by tree edges, and it is a forest embedded in the graph network. In the particular case of lexicon-networks we can also assume that:

- the depth-first forest can be a tree (the DAGW lexicon can be designed to be connected);
- since the network is acyclic when described as sequences of subword units, there are no backwards edges;
- moreover, it is possible to obtain topological sort and to apply DFS in such a way that all cross edges are always from lower numbered to higher numbered states.

These properties means that we can represent the lexicon network as a tree model (just like in Section 10.9) enriched now with forward and cross edges whose destination states have always a higher index.

Observe that the idea of using DFS to obtain a *nearly* topological order is not new. It has been reported, for instance, in [Amengual and Vidal 1998] for a different purpose, namely: to deal with edition operations (see Section 10.4), which is not related to the ideas of storing active states in a consecutive way using a topological order.

Our goal is to extend previous algorithms in such a way that these non-tree edges are tackled as *exceptions* in some way. The benefits of this approach may depend on how a lexicon network resembles a tree.

The forward traversal algorithm of Section 10.9.4 can be easily extended from tree models to more general (DAWGs) lexicon networks in both modalities: when using an auxiliary queue or when traversing the input array of active states with two pointers. In both cases, a priority queue data structure (e.g. a min-heap) is used to store information of children reached by means of forward or cross edges (both of them are tackled in a uniform way since they are simply exceptions that points towards future destinations). Otherwise stated, the min-heap would use the active state identifier as the key to compare values in order to extract the lowest value. The extension of the forward traversal algorithm using an auxiliary queue is depicted in Figure 156, while the corresponding extension for the version using two pointers is shown in Figure 157.

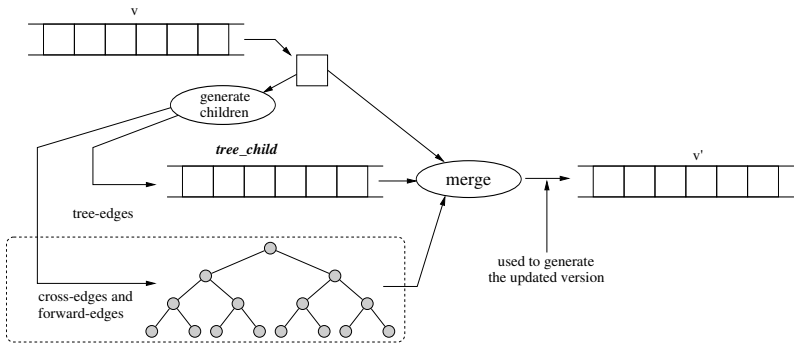


Figure 156: Forward traversal Viterbi algorithm for lexicon networks using sorted arrays and a heap data structure. An auxiliary queue is used to store the tokens associated to the children generated during the traversal by means of tree edges. A min-heap data structure is used to store hypotheses generated by using cross-edges and forward edges. Note that several hypotheses associated to the same lexicon network state can be stored in the heap since there may be an arbitrary number of edges incoming the same lexicon network state (although only one of these edges is a tree-edge).

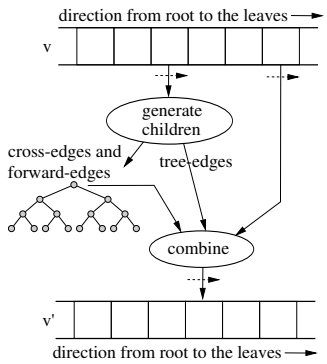


Figure 157: Extension of the forward traversal Viterbi algorithm using pointers. In order to extend this algorithm to general lexicon models, a min-heap data structure is used to store information of children reached by means of forward and cross edges. Each time a destination state is to be created in the original algorithm, elements from the min-heap are extracted (taking into account that the same state may appear several times) and used to activate new states or to combine with the one to be stored.

10.10.2 Adapting the sub-word-centric approach from trees to DAGWs

The extension of the subword-centric approach for tree lexicons described in Section 10.9.8 can also be extended in a quite easy way to the more general case of DAGWs lexicon networks.

Let us remember that this approach consists in jointly processing the active hypotheses associated to the same sub-word model instances. In this way, the processing of each sub-word model instance is performed in the same way as before and, indeed, it could be easily parallelized (although a more fine grained parallelism can be envisaged, it makes sense to assign the processing of sub-word model instances to process elements so, in principle, as much threads as sub-word model types are possible). The only difference between the use of tree models and more general lexicon models is the combination of hypotheses outgoing a lexicon state and incoming to their children. Fortunately, the bucket-sort-inspired approach may be easily applied to this more general case without change. We only have to take into account that a given state can be activated several times.

10.11 SUMMARY AND SOME CONCLUSIONS

reviews include novel proposals

Even the inattentive reader may easily observe that this chapter is quite long in both number of pages and number of sections. What is not so evident is the fact that, besides a quite extensive review of many state of the art techniques, our contributions are not restricted to the last sections where some specialized lexicon decoders are proposed. Indeed, some of the ideas described during the first sections (to the best of our knowledge) are also novel. Let us briefly describe the aims and contributions of each section.

dynamic vs static

We start, already in the introduction, by refreshing the difference between dynamic and static approaches.[77] We have chosen the dynamic approach[78] that can benefit from specialized lexicon decoders that can take advantage of certain lexicon topologies. These specialized algorithms is the main contribution of this chapter. However, we have not limited our study to HMMs since some of the described techniques are suitable for processing segment models (for instance, to convert sub-word graphs into input word DAGs as those used by two-stage decoders).

traditional reviews are restricted to frame sequences

The first section, titled "Segment Estimator Interface" is quite unconventional, meaning that we do not expect to found a similar one in other reviews. The reason is, probably, that other works implicitly limit their scope to decoders processing frame sequences. Since we are dealing with TSGMs, we have to consider the possibility of using DAGs as a point of departure. These DAGs are not the same as those found in multi-stage systems but the ones generated in the first stage of two-stage decoders (their scores are generative likelihoods obtained by segment models), which are the focus of Chapter 11. Moving from sequences to graphs makes it necessary to distinguish between persistent and ephemeral operations. Another interesting question is the possibility of incremental computation, which would become relevant in the case of parallel computation, for instance when using GPUs. Discussions about APIs are also interesting for modularity and flexibility purposes, since different lexicon decoder types can be used in the very same recognizers.

The review of classical Viterbi implementations and usual lexicon organizations provide the necessary background for subsequent sections where novel specialized decoders will take profit of some of these ideas and lexicon organization types. The distinction between propagation and aggregation modalities and the explicit management of a set of active states are essential to understand the decoders proposed in the final sections.

oversegmentation & bundle search

Among the techniques to reduce the cost, we can now pay attention to the use of external oversegmentation and the use of bundle search since they are underestimated or even ignored in many systems and reviews. We have been able to include oversegmentation information in both one-pass and two-stage decoders and some novel oversegmen-

77 Which is a dichotomy restricted to dynamic programming. It would be misleading not to mention that other approaches exists, as is the case of stack decoding which, by the way, probably benefit more from lexicon decoders suited for dynamic than for static DP based systems.

78 We have also explained the reasons. One of them is their greater flexibility.

tation marks have been proposed, although the actual description of these techniques is provided elsewhere (Sections 11.1.4 and 12.4).

jointly decoding several frames

Relating Section 10.6.9 (titled "Other techniques to reduce the cost"), we can mention the proposal of a novel technique that we have called "Jointly decoding several frames" and, although we have not been able to implemented it yet, we are quite confident on its suitability and we have put in our priority list of future work.

detecting OOVs

The detection and emission of OOV words is our first attempt to include in lexicon decoders the capability of emitting the special <UNK> symbol that is often included in LMs to deal with OOVs. We expect that this could be used in overall recognizers to improve their performance when faced with OOVs. In particular, this goes along the line of combining lexicon and lexicon-free recognition systems.

specialized decoders

The introduction of specialized linear lexicon decoders can be seen as a first step towards more practical and significant specialized tree lexicon counterparts. Otherwise stated, this section has been done for pedagogical purposes: the distinction between propagation and aggregation modalities, the explicit management of active states as well as jointly managing multiple hypotheses and jointly processing multiple frames can be concreted and specialized for the particular properties of linear lexicons in a much easier way than for tree or network lexicons. It seems better to introduce them in such a way in order to adapt them later.

Nevertheless, linear lexicons could also be interesting for themselves in some particular scenarios and not only as a preamble for tree and network based lexicons. Some examples of the current use of linear lexicons are detailed in Section 10.5.1 and a novel possible use is a technique that, since we have not seen elsewhere in the literature, we have coined as "sub-word centric lexicon decoding".

Most specialized decoding techniques are based on a topological traversal of the lexicon network in such a way that only active states have to be processed. Some techniques take profit of certain topologies to avoid the use of hashing or similar techniques to locate active states. This is particularly evident for linear lexicons and is adapted later to more general lexicon networks.

forward vs reverse traversals

It is possible to follow a topological or a reversed topological traversal. Each option has its pros and cons:

- forward topological traversal is more appropriate when dealing, for instance, with: 1) null transitions or error correction edition operations, or with 2) jointly processing multiple frames;
- one of the best benefits of the reverse traversal is that tokens incoming to the lexicon decoder are not required until the very end. In this way, most part of the lexicon decoder process can be computed in parallel with the LM and trellis updating procedure.

tree lexicons

Tree lexicons are, no doubt, the most successful and widely spread lexicon decoder type used in most recognition systems. The reason is that they constitute a good trade-off of pros and cons being much simpler than lexicon network decoders which allow higher compactness at the expense of complexity. Section 10.9 is devoted to explain the specialized tree lexicon decoders that we have designed during this

PhD work. Both the destructive updating and the persistent modalities are considered although, when dealing with destructive updating, we compare our system with a technique, called Active Envelope, which is the one that most resembles our proposal in the previous literature.[79] The differences with this technique are described.

Let us remark that, although some of these ideas and experimental results have been published in [España-Boquera *et al.* 2007], although many improvements and further developments have been first published in this PhD report.

Some variants of the proposed tree lexicon decoders exist. They may differ in the underlying HMM topology (e.g. specialized for left-to-right HMMs, for Bakis topologies, and even for the more restricted case where all sub-word units are modeled by left-to-right models with the same length) although the use of general HMM units, even with several entry/exit points, has been contemplated. The extension for jointly managing multiple hypotheses is achieved by means of a double ordering of active states, which is a novel idea to implement the bundle search. This has been crucial in the implementation of the one-pass recognizer of Section 12.3 that, together with the use of the LM interface (with a bunch mode) from Section 9.1 allows a novel recognition system that can be parallelized by assigning a different thread to each internal component (lexicon decoder, LM, etc.) while achieving the remarkable property of not requiring the use of dictionary search techniques (e.g. hashing) in any part.

double ordering

Besides, these algorithms are friendly with the cache due to the pattern of memory accesses. These algorithms could be further improved by means of cache pre-fetching.[80] These algorithms can be generally considered cache-oblivious because they do not need to know the size of the cache.

The section on tree lexicons is not only limited to decoders for processing frame sequences. Besides the fact that persistent versions could be easily used with input DAGs, an extension to deal with DAGs labeled with sub-word units is capable of performing error correction decoding. The forward topological traversal is used in this case to implement the case of deletions in an efficient way. These error correcting DAG decoders are intended to be used in the decoders, described in in Section 11.2, for generating a word DAG from a sub-word one.

The chapter finishes with some completely novel approaches to work with general (acyclic) lexicon networks whose implementations (extending our tree lexicon decoders) constitute one of our imperative future work assignment related to this chapter.

novel general lexicon network decoders

79 We were aware of the active envelope algorithm much later than our designs were conceived and implemented.

80 This technique consists in forcing memory pages to be loaded into the cache, without having to wait for them, in order to reduce the latency when this memory page is shortly required.

11 DAG GENERATION

Each generation imagines itself
to be more intelligent than the one
that went before it, and wiser
than the one that comes after it.

George Orwell

CONTENTS

This chapter studies the algorithmic and implementation details of the first stage of two stage decoders described in Chapter 4. The goal is to process the input signal (a frame sequence) in order to generate a DAG whose edges are labeled with segment generative likelihoods. Nevertheless, other modules which generate the same type of DAGs, but receive as input one or several DAGs, are also described.

More concretely, we are talking about some dataflow components, presented in Section 8.3.2, that we coined, respectively, `seq_graph_gen` and `graph_graph_gen`. Some examples of decoding architectures using these components were proposed in Section 8.4.

It is important not to confuse the DAGs discussed in this chapter with the more known word graphs used to compactly represent results from a recognition stage. These more common graphs were introduced in Section 3.1.15 and, even if the graphs proposed now can be represented by means of the same data structures, they are produced and have to be interpreted differently: the more common and previously described word graphs contain the contribution of the LM and what is relevant in them is the overall score of paths rather than the score of individual segments. Let us remark that it is possible, nevertheless, to obtain a DAG with segment generative likelihoods as a *by product* of a former recognition stage (as illustrated in Figure 110).

related to time-conditioned decoders

As explained in Section 10.2.3, the graph generation of two-stage decoders is very related to the time-conditioned decoding approach [Ortmanns and Ney 2000]. In both cases, words hypotheses inside lexicon decoders have started at the same input position. Although two-stage decoders are more general, they are essentially equivalent when both sub-word models and LMs are constrained to be regular, as is the case of HMMs and n-grams, respectively.

transducer composition

Let us also remark that the generation of the word DAG, either from the input signal or from another sub-word DAG, can be seen, in most cases, just a particular case of transducer composition. In this regard, and as with lexicon decoder estimators from the previous chapter, the modules described now are circumscribed to the dynamic decoding approach since they are specialized algorithms that does not need to be detailed in the static composition counterpart. However, let us stand up for the dynamic approach by remembering that it is possible to go beyond those regular models, in DAG generators, when segment estimators are not limited to HMMs.

incidence serialization protocol

Since we try to implement the DAG generators as dataflow components, we assume that the reader is familiar with the DAG incidence serialization protocol of Section 8.2. This is the main reason why the algorithms of this chapter are expressed in a reactive programming style which consists in specifying which messages are sent and how the internal state is updated at some particular events. More concretely, both `seq_graph_gen` and `graph_graph_gen` are filters in the sense that they have an input and an output port.

This chapter is mainly divided into three sections, each one describing a different dataflow component differing in the input modality:

a) a frame sequence. Optionally, there can be over-segmentation marks associated to frame boundaries. This component can make use of the specialized decoders described in Chapter 10;

b) another DAG like those generated in this chapter. The most obvious example is the generation of a word DAG from another DAG labeled with sub-word labels (e.g. graphemes, phonemes,...);

c) it is also possible to combine and filter one or several DAGs like those generated in this chapter. This can be used to implement, for instance, multi-modal and multi-stream recognizers, as described in Section 8.4.6. The idea is to combine DAGs obtained from different inputs corresponding to the same word sequence. The advantage is that the subsequent decoder remains the same.

feedback

A noteworthy feature is the possibility of receiving feedback information from *any* decoder which accepts the corresponding serialization protocol as input (an example illustrating the use of the feedback channel is found in Section 8.4.3). This feature allows modularity *without losing one of the main advantages of a monolithic implementations: pruning based on all information sources*.

top down filtering

The proposed algorithms perform top down filtering instead of a naive bottom-up approach that would deal with all possible spans of the signal no matter if they are reachable from the initial vertex or able to reach the final one. This makes them unsuitable for some word-spotting applications.

A NOTE ON THE DATAFLOW IMPLEMENTATION

push vs pull

The implementation of dataflow components discussed in Chapter 8 distinguished between pull and push architectures: In push architectures, the sources send tokens which cause other pieces to react and send more tokens. In pull architectures, the sink components demand new data which causes other components to also demand more data until sources are reached.

The dataflow implementation of our `April` toolkit is based on the push alternative which means that the components are specified by the following pieces of information:

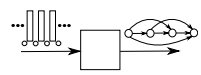
- how the internal state is represented and how it is initialized;
- which actions are to be taken when receiving each possible type of input token from the input channels. This also depends, obviously, on the internal state. These actions, in turn, basically consist in updating the internal state and sending output tokens by the output channels.

Besides, as stated in Section 8.1, the current implementation is based, for the sake of portability, on cooperative threading: each dataflow module explicitly yields the control to the others leading to a negligible overhead w.r.t. non-dataflow implementations of the same designs. Each dataflow component can be structured over a finite state model skeleton. Indeed, dataflow components are C++ objects that inherit a basic **`run`** method which usually consists in a **`switch`** statement controlled by a FSA state attribute. In this way, each time a token is received, the code associated to the current state is activated. This code is responsible for checking the token type and performing the required actions, including the delivery of outgoing tokens. Dataflow components ready to execute **`run`** are stored in a queue while others attend events to become ready. When a component receives tokens by some of its input channels, it may become ready. Although this is compatible with the use of a pool of threads, a pure thread based implementation would have lead to a more classical sequential description of the corresponding algorithms at the expense of more overhead.[1] This is the reason why some algorithms are depicted as flowcharts.

flowcharts

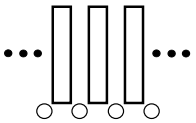
11.1 FROM A SEQUENCE (seq_graph_gen)

The `seq_graph_gen` dataflow component is a filter: it is connected to a previous component by an input channel, which receives frames, and to another dataflow component (usually a decoder) by an output channel, where a DAG serialized with the incidence protocol is transmitted. An additional input channel from the decoder back to the `seq_graph_gen` will be optionally included in the extension proposed in Section 11.1.3, but let us first introduce a basic model in order to gradually incorporate more capabilities to it.



1 Many concurrent software architectures rely, for efficiency, on lightweight threads which usually come in several flavors depending on the language, operating system and available libraries. We designed our particular solution with similar purposes.

11.1.1 Basic model



Although the information received by the input channel is a frame sequence, we will assume that the frame source will also emit tokens representing frame boundaries.[2] The motivation for including these tokens is twofold: firstly, they may simplify the implementation and, secondly, they constitute a preadaptation to the extension for dealing with over-segmentation proposed in Section 11.1.4. The input sequence is also finished by a special `signal_end` message to notify the end of the sequence. This is relevant to determine the repertoire of expected input token types. Relating the output:

- vertices of the emitted DAG correspond, in principle, to frame boundaries, but this will be later generalized when including context dependent units or when using a control automaton;
- edges are labeled with lists of tuples[3] comprising, each one, a word identifier and its associated generative likelihood. This list is computed by segment or lexicon estimators as those described in Chapter 10.

In order to implement that, we need a pool of lexicon decoders: each decoder is associated to a different time start (or frame boundary) and is devoted to process word/segment hypotheses started at this precise instant, as illustrated in Figure 158.

main loop

In order to process the frame sequence in an online way, a vertex is serialized as soon as the corresponding frame boundary token is received. This is very easy when using the incidence protocol since all its incoming edges are already known. After emitting all the edges, the `no_more_in_edges` message is sent. Indeed, we designed this protocol with these dataflow components in mind. The `no_more_out_edges` message is not sent in this version since it is related to pruning (described in the next subsection). In case at least one word has finished at this frame boundary, a new decoder associated to this new time start in created an included in the pool. When a frame token is received, it is used to update the word hypotheses of all decoders, as depicted in Figure 158.

The previous paragraph described the main loop of the component. Besides that, some initial and final tokens of the incidence protocol have also to be emitted: `begin_dag` at the beginning and `end_dag` token at the end. This event is detected at the input because a `signal_end` token is received by the input channel. Besides, the `is_initial` and `is_final` flags have to be included as attributes to the initial and final output vertices, respectively. The `is_final` flag poses more problems since the output vertex should have delayed until knowing if the next token is `signal_end` or not.

2 Observe that, although the input is also a DAG (since sequences are obviously DAGs), it is not sent using the incidence protocol. Using this protocol leads to the module `graph_graph_gen` described in Section 11.2 where it will be clear that the current `seq_graph_gen` is just a particular case. In spite of that, we believe that it is worth implementing the `seq_graph_gen` to take profit of the fact that sequences allows the use of persistent (or *in place* updating) lexicon decoders and to allow a simpler interface with the frame source generator.

3 Labeling edges with lists is just a way to represent a non-simple graph as a simple one, where *simple* means that at most one edge connects two given vertices.

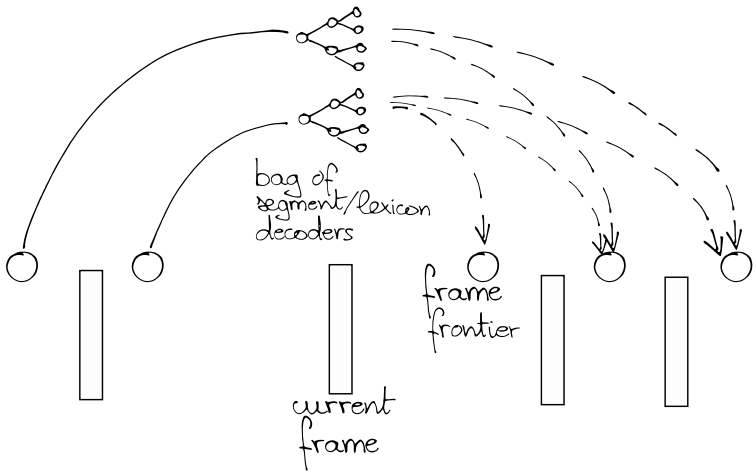


Figure 158: Scheme illustrating the internals of `seq_graph_gen` with a bag or pool of lexicon decoders associated to time starts. This pool is updated when an input frame is received. Frame boundaries are associated to output DAG vertices. In order to implement a quite naive top-down filtering, a new decoder is included in the pool, at a frame boundary, only when at least some word is generated at this point, by an already existing decoder, and emitted afterwards.

A dataflow illustrating the structure of this basic version is depicted in Algorithm 14. This flowchart is slightly more complex than expected since we have considered the possibility of empty sequences where only a `signal_end` token is received and due to the fact that we have to delay the output to determine if the vertex to be emitted is final. The processes shown in this flowchart are described in Algorithm 15.

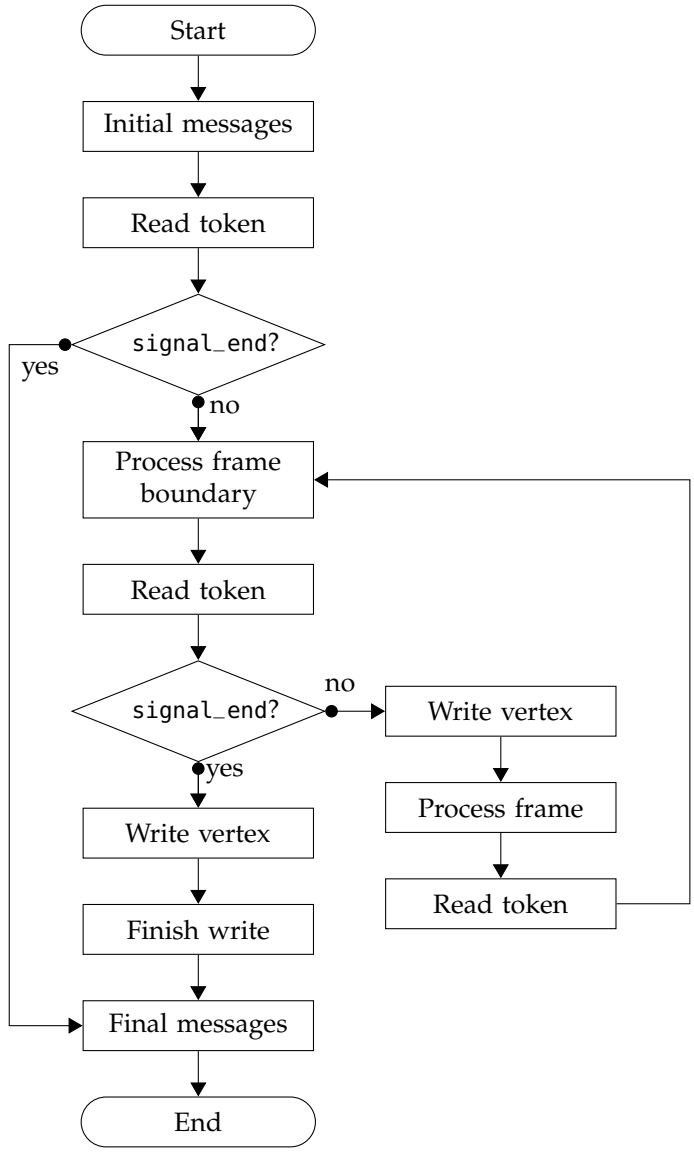
Relating the implementation of the pool of lexicon decoders, two basic choices have been considered:

- a different segment/lexicon estimator is associated to each start time. In this case the pool is implemented as a dictionary where keys are frame boundary identifiers and values are decoder instances;
- a sole segment/lexicon estimator where hypotheses associated to different time starts may coexist. These hypotheses are internally represented by means of tokens where the time start is the token field used to distinguish which hypotheses must compete between them, see Chapter 10 for details.

Beside the pool of lexicon decoders, the internal data structure for implementing the `seq_graph_gen` includes:

- a variable representing the current state of the underlying FSA skeleton and a counter to enumerate the frames that have been received so far. This counter is used to generate DAG vertex identifiers, as described in Algorithm 15;
- a data structure for storing the tuples computed by the lexicon decoders before sending them by means of `edge` tokens.[4]

4 This data structure is not mandatory in the basic version of the algorithm but is convenient when including the control automaton.



Algorithm 14: Flowchart of the basic `seq_graph_gen`.

```
Method initialMessages()
    vertexCounter ⟵ 0
    send(begin_dag)
    return
Method processFrameBoundary()
    send(vertex (vertexCounter))
    if vertexCounter == 0 then
        send(is_initial (vertexCounter))
    return
Method writeVertex()
    vertexCounter ⟵ 0 vertexCounter +1
    forall the x ∈ pool of edges do
        send(edge (x))
    send(no_more_in_edges))
    return
Method processFrame(frame)
    forall the decoder ∈ pool of lexicon estimators do
        stepViterbi(decoder)
        /* the results are collected in an pool of edges */
    return
Method finishWrite()
    send(is_final (vertexCounter))
    return
Method finalMessages()
    send(end_dag)
    return
```

Algorithm 15: Basic `seq_graph_gen` algorithm.

The cost of processing a new frame is dominated by the loop which updates the pool of segment/lexicon estimators. Note that the number of these estimators grows linearly with the size of the frame sequence. The output DAG would become complete, leading to a quadratic cost, unless pruning techniques were applied.[5] The lack of pruning also explains why this basic version never emits `no_more_out_edges` tokens, which are associated to the incidence protocol and used to indicate that a given vertex will no longer be used as origin of edges. These issues will be addressed in the following sub-section.

Observe that the same `seq_graph_gen` would emit a DAG based on words or on sub-word units depending on the models used in the pool. The generation of a DAG of sub-word models makes sense either when using a sub-word based LM or when the output is connected to a `graph_graph_gen` used to obtain a word DAG afterwards. Let us also remark that nothing prevents us from using segment estimators other than HMMs.

These decoders would simply use the same decoder interface (Section 10.2) although they would store the frames in order to delay the computation of outputs on demand. A notable example showing that these segment based techniques can be used at the word level is the use of holistic techniques to deal with frequently miss-recognized words. Some quite promising results in this respect are reported in Section 15.5.

11.1.2 Pruning

Some kind of pruning strategies must be considered in order to make the `seq_graph_gen` component practical: the joint combination of beam search and histogram pruning of Section 10.6.1 has been chosen to this end.

Although is suffices to use lexicon estimators implementing these pruning criteria, it is important to use the same criterion/threshold in all the segment estimators of the pool despite of the fact that they have started at different instants. This can be properly achieved by initializing each lexicon decoder with the best score of the pool at this moment.[6] This initial score must be stored in order to reuse it later to normalize the hypotheses emitted later by the decoder.

The idea of pruning all hypotheses, no matter at which time-start they belong, is not new [Oerder and Ney 1993]:

> *Pruning relative to best path. The trees must not be pruned independently but relative to the current overall best score (found in any tree). Since every tree is started with the current best word ending score, this results in a consistent and very efficient pruning.*

5 A complete DAG in the sense that every vertex is connected to the preceding ones, so the number of edges grows quadratically. Observe that, although the number of edges grows quadratically, the amortized cost of processing each edge is constant in average, when using HMMs, since each Viterbi step takes profit of previous steps applied over the prefixes. This is not the case of certain segment models which would lead to an overall cubic cost.

6 This will change when including the feedback channel in the next subsection.

The basic implementation of the `seq_graph_gen` component from previous section must be slightly modified in order to not only include this initial score but also detect when all hypotheses from a given lexicon estimator from the pool are no longer alive. Since these decoders are not re-entrant, when a decoder becomes completely deactivated it will remain in this state forever. This situation has to be detected for two reasons: 1) to release the lexicon decoder resources, and also 2) to notify this fact to the dataflow component receiving the output DAG. This is the role of the `no_more_out_edges` token. The receiver of output tokens takes profit of this message, in turn, to release its own resources.[7] The required modifications of the component are illustrated in Algorithm 16.

```
Method constructor()
    bestScore ⟵ 0
    /* the rest of the code is unchanged */
    return
Method writeVertex()
    vertexCounter ⟵ 0 vertexCounter +1
    forall the x ∈ pool of edges do
        send(edge (x))
    send(no_more_in_edges))
    return
Method processFrame(frame)
    forall the decoder ∈ pool of lexicon estimators do
        stepViterbi(decoder)
        /* viterbi decoders make use and modify bestScore */
        /* the results are collected in an pool of edges */
    return
```

Algorithm 16: Modifications of `seq_graph_gen` to include pruning.

Relating the cost, although it remains quadratic in the worst case, the number of active lexicon models is proportional, in practice, to the average duration[8] of words, making the cost of generating the DAG, in practice, linear with the input size. This linear cost can also be achieved by establishing an upper bound on the duration of segments in TSGMs, as discussed in Section 4.2.1.

11.1.3 Using feedback information

The feedback information is a novel technique proposed to improve the performance of pruning. The problem of semi-decoupled architectures is that the DAG generation module lacks some information that is only available by the decoder using this DAG. It is possible,

7 This issue is addressed later depending on the dataflow component connected to the output: `graph_graph_gen` in Section 11.2 and the DAG decoder in Section 12.2.

8 Observe that this is the same property found in time-conditioned decoders, as pointed out by [Ortmanns and Ney 2000; Section B].

overcoming limitations

for instance, that some word hypotheses with a high likelihood, from the lexicon generator point of view, become worse when taking the LM information into account and vice-versa. The inclusion of an additional feedback channel, explained in Section 8.4.3, allows dataflow architectures to overcome this limitation apparently inherent to semi-decoupled architectures.

The basic idea is that, when the DAG decoder processes all the edges directed to the new DAG vertex, it sends back to the DAG generator the best score, associated to this vertex, of its own trellis. It is used to initialize the new lexicon decoder starting at this new frame boundary instead of using the best score of the pool. This is an upper bound of the best usage this lexicon decoder would have in the integrated version of the algorithm. As with the previous case, this score is removed from the scores of output word hypotheses since it only serves for pruning purposes.

This feature is included by default since a unique `seq_graph_gen` implementation can detect if the feedback input channel is connected or not and adapt its behavior accordingly. Making the feedback channel optional makes sense since there are practical recognizer examples with and without feedback, as shown in Section 8.4. The modifications required to implement this `seq_graph_gen` extension can be found in Algorithm 17. The basic modification lies in the read token feedback method that overwrites the **`bestScore`** attribute.

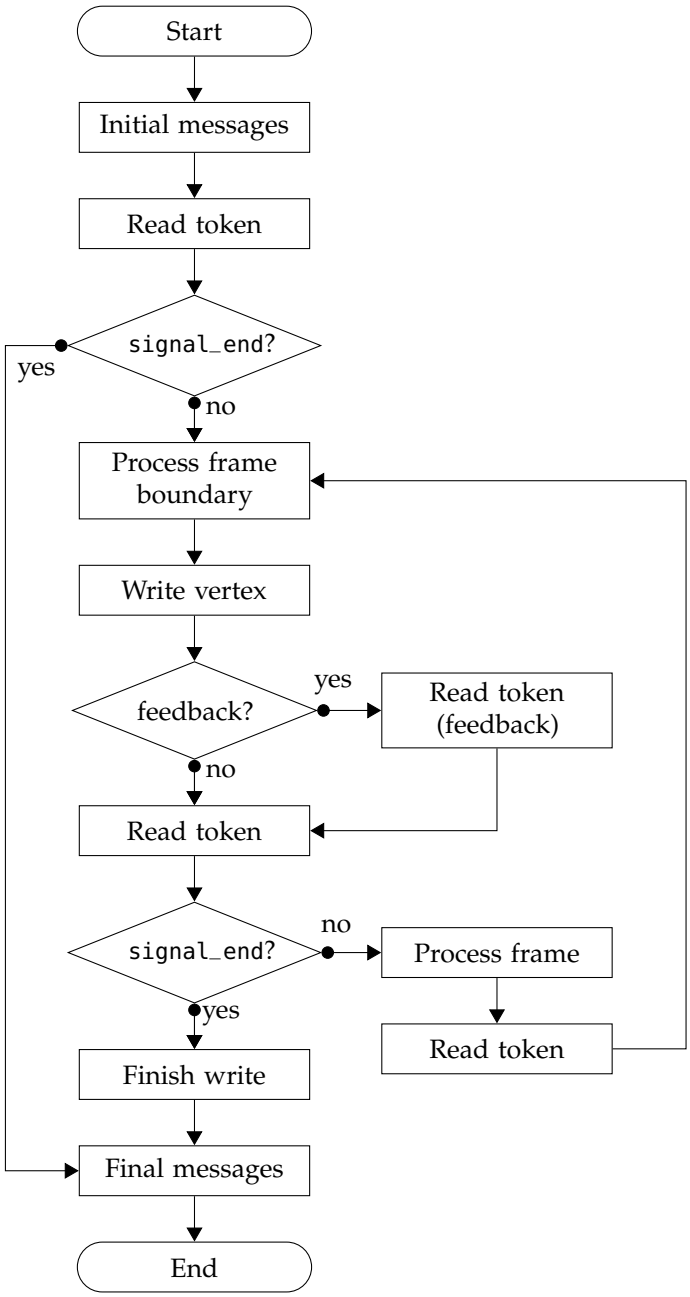
Note that this feedback only makes sense because of the incremental generation of DAGs and the fact that they are used while they are being generated. Let us remark that coupling the DAG generation by means of the serialization protocol has, obviously, other pros and cons:

- on the one side, the DAG has not to be stored as a DAG and it is not necessary to wait until the end of the sequence to begin the decoding procedure, either;
- on the other side, serialization and feedback constraint the parallelization of this component.[9] Additionally, if the entire DAG was generated before sending it to the DAG consumer, some dead-end paths could be removed.

Note that, by construction, all vertices and edges are reachable from the initial vertex but, because of the pruning, they are not necessarily useful (there is not always a path from them to the final vertex). Removing the non-useful part of the graph can be done very efficiently, but requires to wait for the DAG to be completed. The problem of this delay is that the feedback can no longer be used and this implies a less accurate pruning. In this regard, we can mention the possibility of using a first decoder with a cheaper (e.g. lower order n-gram) LM to generate a first DAG to be used with a more costly LM afterwards (by using the first one to prune the input DAG instead of using the output trellis of the first decoder, as explained in Section 8.4.4).[10]

9 A naive bottom-up DAG generator, without the serialization constraint, could be implemented in an embarrassingly parallel way, see `https://en.wikipedia.org/wiki/Embarrassingly_parallel`.

10 This is related with the approach described in [Ortmanns *et al.* 1999] where a time-conditioned word graph is constructed using a LM and used, perhaps, with a different LM afterwards.



Algorithm 17: Flowchart of `seq_graph_gen` with feedback channel.

delaying feedback

Let us see how it is possible to support some delay in the feedback channel. In this way, the `seq_graph_gen` module does not need to wait after emitting the `no_more_in_edges` message. This feature is not required in a one-thread setting since the cpu is not idle while waiting for the feedback token, but it can be convenient in multi-core and in distributed dataflow implementations. Moreover, the possibility of delaying the emission allows to prune some of the dead hypotheses. In order to delay the feedback, it suffices to locate, in the pool, the lexicon decoder of the vertex mentioned in the feedback message and to modify the score of their hypotheses by adding the proper offset to them. When the pool is implemented by using just one decoder containing tokens associated to different time starts, it is possible to traverse the token vector modifying the scores of the tokens whose time start matches the one we want to adjust. This would essentially lead to pruning the same hypotheses as if the feedback message arrived just before the creation of this decoder.

richer feedback information

It is possible to go beyond this basic feedback strategy where the feedback information is limited to the best usage of the scores generated by the lexicon decoder starting at a given moment. One of the ways to extend this feedback consists in using a lattice of predetermined lexicon decoders as described several times through this work.[11] The DAG decoder component responsible for sending the feedback token would not only compute the best score associated to hypotheses reaching a given input vertex, but also the most specific/-suitable type of lexicon decoder type (from a pre-determined subset) to generate the word hypotheses starting at this frame boundary. Since this is computed by the DAG decoder, it will be detailed in Section 12.2 (in the subsection associated to the `no_more_in_edges` message).

11.1.4 External over-segmentation

Although the possibility of enriching the input with additional over-segmentation information was first discussed in Section 3.1.4, let us briefly remember the different types of marks that can be associated to frame boundaries by an external over-segmenter:

NON-SURE FRONTIER means that a word may finish or start at this point;

SURE NO-FRONTIER indicates that no segment is allowed to finish or to start in this moment;

SURE FRONTIER is a boundary which must belong to any possible segmentation; and finally

FAT SURE FRONTIER is an extension of the previous case, proposed in this work, where several consecutive frame boundaries act as a global sure frontier. This means that any allowed segmentation must use at least one of the frame boundaries of this group, as illustrated in Figure 24 (replicated as Figure 159 for the sake of convenience).

11 It has been described, at least, in Sections 6.10.1, 9.1.2 and in a subsection of Section 10.6.7 described in this work: titled "LMLA for 2-stage and time-conditioned decoders".

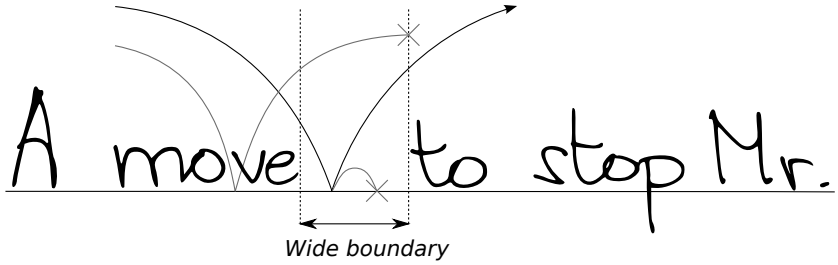


Figure 159: Example of a *fat sure frontier*, (this figure is a copy of Figure 24).

why this feature is not widespread

Over-segmentation is not commonly used with HMM-based (or related) decoders since it is not reliable in ASR tasks (which have, undoubtedly, lead the development of this technology). It is not surprising that this technique has not been applied in HTR either, since *the application of HMMs in HTR has inherited the tradition and techniques from ASR*. Nevertheless, we have experimentally shown that it is possible to apply over-segmentation to some HTR tasks, as reported in Chapters 14 and 15, so it would be a pity to ignore this technique.[12]

In any case, the idea of reducing the positions at which a vertex can be started is not new, although it seems limited to the use of "Sparse word start points" viewed as a pruning heuristic and not related to the use of external over-segmenters [Oerder and Ney 1993]:

> *Sparse word start points. A further significant reduction in search effort can be achieved by not allowing word startups at every frame, but only every other or every third frame. This results actually in a small increase in error rate, but in the graph framework this increase can easily be compensated for by tuning the pruning threshold.*

This could be easily included in our setting by using a trivial over-segmenter (available in our `April` toolkit) which emits sure no-frontiers interleaved with non-sure frontiers every n frames, being n a parameter to control the periodicity.

pre-adapted

The basic `seq_graph_gen` can be very easily extended to deal with over-segmentation marks because we have already assumed, in the basic scheme of Section 11.1.1, the existence of specific input tokens representing frame boundaries. Now, it would be enough to include an attribute in these tokens indicating the type of over-segmentation mark and, of course, to use these marks properly. Relating fat sure frontiers, since they last several consecutive frames, they can be represented in the frame boundary tokens by means of the following labels: "begin group", "middle group" (repeated as many times as necessary in the middle part) and, finally, "end group".

emulate lack of over-segmentation

We have decided to include the over-segmentation extension in the default implementation since there is no gain in creating an implementation lacking this feature: it suffices to include always the mark "non-sure frontier" in order to emulate the basic behavior.

12 As stated in Section 1.1, we try to show, in this work, that what some people coin "segmentation free" and "explicit segmentation" can be extreme points of a continuum rather than a dichotomy.

Table 7: Over-segmenter types expressed as a combination of features.

Frontier type	Generate output	Generate marked	Mark	Remove marked
sure frontier		✓	✓	✓
sure no frontier				
no sure frontier	✓			
begin group		✓	✓	
middle group		✓		
end group		✓		✓

In order to better factorize the required behavior, we have designed the following boolean indicators that are applied in the same order that they are defined:

GENERATE OUTPUT is used to indicate that all the segment estimators from the pool are asked for output values at this point;

MARK means that all the segment estimators from the pool are marked;

GENERATE MARKED is the same as "generate output" but only takes into account the lexicon decoders that have been marked;

REMOVE MARKED indicates that all marked segment estimators have to be removed.

In this way, the different over-segmentation marks are associated to or expressed as a combination of these flags, as shown in Table 7. The details of the corresponding procedures, can be summarized as follows:

- the "mark" flag is applied to the elements of the pool before the new decoders starting at the current boundary are created;
- "remove marked" is applied after "mark" so that a sure frontier can use both flags simultaneously;
- relating fat sure frontiers, the idea is to mark all decoders created before the frontier, avoid the generation of words whose span is completely inside this group and, finally, forbid any word hypothesis to cross the frontier (by removing decoders after closing the fat boundary).

  The purpose of distinguishing between "generate output" and "generate marked" is to prevent the emission of short words inside the fat frontier, as the short segment depicted in red inside the wide boundary of Figure 159;
- observe that a sure-frontier behaves like a fat sure frontier of size just one frame boundary: it suffices so consider the joint behavior (the logical *or* operator) of flags associated to "begin group" and "end group" marks.

Needless to say, the pool representation has to be modified to associate a mark boolean flag to each lexicon decoder instance.

11.1.5 A control automaton and across-word context dependency

As mentioned in the introduction of this chapter, the behavior of `seq_graph_gen` can be easily explained in terms of transducer composition: the frame sequence (where frame boundaries are states and frames label edges) is composed with the simple one-state automaton depicted in Figure 160. Observe that this is not an stochastic FSA even if the lexicon network used in the loop holds this property. Let us also remark that the `seq_graph_gen` versions explained up until now made use of lexicon network decoders. However, there is nothing equivalent, in the `seq_graph_gen` descriptions viewed so far, to an automaton like the one depicted in Figure 160 to control the behavior. Indeed, the fact that this *by default* automaton has only one state makes it very easy to omit it. However, there are situations which motivates the use of non-trivial automata and the need to include them in the `seq_graph_gen` dataflow component. Let us put two different examples: 1) How can initial and final special silence units be included? and 2) How can context dependent units be taken into account? Instead of generating an ad hoc version of the algorithm to include these features, we prefer to describe a general extension that mimics the transducer composition approach where these features can be expressed in a straightforward way.

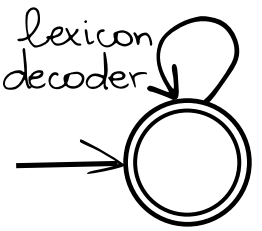


Figure 160: One-state automaton to control `seq_graph_gen` component.

A common point of view of transducer composition is the state-pair approach[13] reviewed in Section 5.5.1. Although DAG vertices have been associated, up to now, to frame boundaries, the state-pair approach indicates that they are better described as tuples of the form ⟨frame boundary, automaton state⟩. Since the trivial FSA discussed before had only one state, the second component of the tuple has been ignored for the moment. But the example of an initial and an optional special silence model, depicted in Figure 161, makes the use of these tuples necessary. We can observe, from this figure, that several lexicon decoder types are used simultaneously in the same `seq_graph_gen` instance. We have assumed a Mealy[14] control automaton, meaning that labels (indicating the type of lexicon decoder used at the edges) are placed in the arcs. Since several transitions may be labeled with the same lexicon decoder type, it makes not sense to evaluate more than one instance of each type of decoder when starting from the same frame boundary.

state-pair approach

We have also allowed null transitions in the control automaton. This makes also convenient to be able to emit tokens representing null edges (already contemplated by the incidence serialization protocol) which, in turn, would force the subsequent dataflow components (e.g.

13 Transducer composition has been thoroughly addressed in Section 5.5. One of the contributions of this work is to propose a point of view different from the state-pair approach that works well with null transitions without the use of the filter composition trick. However, we do not need to take this extension into account now since at least one of the automata (the input data) lacks null transitions.

14 Regarding the Mealy vs Moore confrontation, we are in favor of using the Mealy approach (remember the "egg" from Gulliver's Travels depicted in Section 5.4.3).

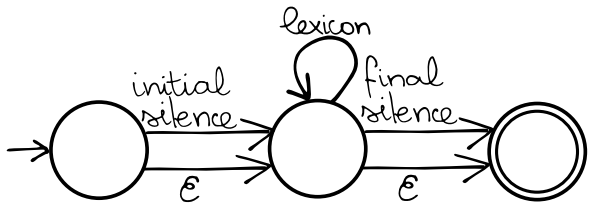


Figure 161: Control automaton for `seq_graph_gen` with optional initial and final silences.

`graph_graph_gen`, DAG combinator and DAG decoder) to accept DAGs containing these null transitions. In any case, this extension offers a great flexibility in several situations which, in our opinion, has made this extension worth implementing.

The inclusion of a control automaton makes the implementation of the `seq_graph_gen` module more complex. In particular, the pool of lexicon decoders is no longer indexed by the frame boundary (or time start) but also by the lexicon decoder type (several decoder types can be activated at the same time and coexist). Each lexicon decoder type has a set of active instances associated, each one, to a subset of previous frame boundaries and accompanied by an initial score[15] and also by a list comprising those arcs of the control automaton that are using this decoder type.[16] The implementation choice of lexicon decoders where a sole decoder instance contains hypotheses associated to different time starts is naturally adapted to the version where there is a sole instance for each decoder type.

The most relevant changes in the `seq_graph_gen` implementation, take place in the following procedures:

WRITE VERTEX now, several vertices can be emitted at the same frame boundary. The order these vertices are emitted is not arbitrary since there can be null transitions between them. This order can be computed beforehand by performing a topological sort of the states relating null transitions (loops composed of null transitions are not allowed). Let us also remark that, since the incidence protocol requires vertices to be identified by natural numbers, a dictionary is used to map those tuples to consecutive integer values;

PROCESS FRAME do not suffer significant changes: the set of lexicon decoders of the pool are updated with a Viterbi step using the current frame;

FINISH WRITE must be slightly modified since only those DAG vertices associated to a final automaton state have to be marked as finals (by sending the corresponding `is_final` token).

15 Used to improve pruning and stored to normalize the output. Since this score can now be related to several output DAG vertices, it may be computed from several feedback tokens as their maximum value. The extension of feedback from scores to lexicon types also requires that the `seq_graph_gen` module is able to operate with the lattice of lexicon decoder types.

16 This list contains a (not necessarily proper) subset of the arcs that are labeled with the corresponding lexicon decoder type.

Dealing with silences in ASR and spaces in HTR

The role of silences in ASR and spaces in HTR can be probably generalized to other related tasks. Although they are not always optional in the same way,[17] they have in common that are not necessarily handled by the LM. Some authors[18] deal with them in a curious way: by creating a lexicon description where every entry has an optional silence at the end. This approach is not only sub-optimal, since the lexicon tree would replicate these silences, but would also affect the posteriors and end time boundaries associated to these words. This is the reason why the use of null transitions has been proposed to deal with silences in the word graphs. The problem of these redundant silences when computing posteriors is discussed in [Wessel 2002; Section 4.2.2.3].

It is possible to design a control automaton, an extension of the one proposed in Figure 161, where:

- different type of silence models are used at the beginning and end of the utterance, on the one side, and between words, on the other (as is usually done in many systems reported in the literature);
- it is trivial to associate the code of the null transition to the hypotheses emitted by these silences models; and, interestingly,
- it is also straightforward to design the control automaton in such a way that consecutive spaces are always forbidden.[19]

In this way, the control automaton may simplify dealing with optional silences or spaces.

Across-word context dependency

The use of within word context dependent units is transparent from the `seq_graph_gen` point of view: it suffices to use context dependent units inside the lexicon decoder. However, modeling across-word context dependency requires an special handling which is similar to the use of context dependent units when generating a DAG of sub-word units, as depicted in Figure 162 *a)* and *b)*. We can observe, from this figure, that there can be several vertices associated to the same segment boundary, just as when using a control automaton. Indeed, it is possible to model the generation of context dependent sub-word units by means of such an automaton, which would be similar to the well known "context-dependency transducers" [Riley *et al.* 1997; Rybach *et al.* 2013]. These transducers perform a mapping form any arbitrary sequence of context independent sub-word units to their corresponding context dependent version. They have been commonly used in the transducer composition approach to ASR. In a similar way, although context-dependent sub-word models where used to generate

context dependency transducers

17 Continuous speech may lack spaces between words, but many handwriting systems require a proper separation of words.

18 We have seen, for instance, many HTK recipes working in this way.

19 This would not necessarily prevent the existence of consecutive null transitions in the DAG since, besides silences, the control automaton can have null transitions that are usually converted into null DAG edges.

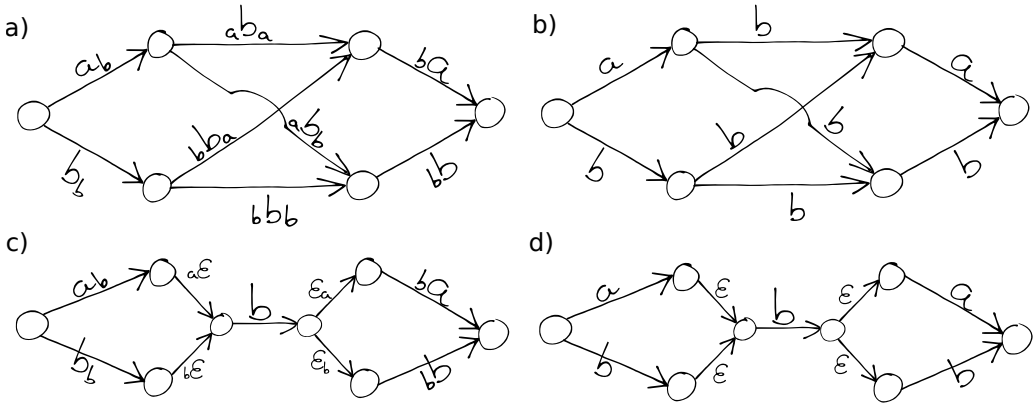


Figure 162: Example of a DAG with context dependent sub-word units: (a) the DAG labeled with the actual context dependent units used for computing the segment likelihoods. We can observe that several vertices (vertically aligned) are associated to the same input signal position, in (b) the actual DAG sent by the `seq_graph_gen` module where edges have context independent labels (scores are not depicted to simplify the picture), (c) a possible way to factorize the different types of context dependent units associated to the same central span of the DAG and, finally, (d) the previous DAG as emitted at the output of the `seq_graph_gen` (with context independent edge labels).

the context-dependent sub-word DAG, the DAG edges would be labeled in a context-independent way so that the subsequent dataflow components[20] do not need to be aware of context dependent models.

The generation of across word context dependent DAGs can also be easily modeled by means of a control automaton. Unfortunately, the naive or direct application of this technique is quite inefficient because each lexicon decoder instance would have to be replicated, at each activated frame boundary, for each possible phonetic context type.

This issue has been addressed in other across-word decoder types (e.g. one pass decoders), when using a tree lexicon decoder, by observing that the lexicon decoders differing in the left context of the initial sub-word models may share the greater part of their structure[21] so that, instead of using a different model for each possible context, it is better to have a sole decoder with several entry points, as explained in Section 10.2.4. The connections from the entry points to the common parts of the rest of the lexicon network can be modeled, in the output DAG, by means of null transitions, as depicted in Figure 162 *c)* and *d)*.

heuristic approach

Another possibility more compact and simpler to implement, although heuristic and approximate, consists in activating all the required across-word contexts while avoiding to distinguish these context in the DAG vertices: using always the best score no matter the fact that the right context of an edge would mismatch the left context of the next one.

20 The most adequate being the `graph_graph_gen` described in Section 11.2.

21 The different phonetic contexts lead to the same states after a few arcs, their number depending on the context dependency lengths (triphones, quinphones, etc.).

Relating previous work that can be found in the literature on the use of across-word modeling when using the kind of DAGs discussed in this Chapter,[22] the most classical two stage decoding approaches (as discussed in Chapter 4) where based either on non-probabilistic distance-based templates or on segment models which, to the best of our knowledge, did not make use of context dependent units, not to mention across-word context dependency. The most similar thing that we have found in the literature is the use of across-word context dependency in time-conditioned search [Nolden *et al.* 2010; Section 3.4] and the generation of time-conditioned word graphs with cross-word triphones [Ortmanns *et al.* 1999; Section 3]. In both cases, there is a time-conditioned one-pass decoder which integrates LM information. The creation of what they call "time-conditioned word graphs" is viewed as a *by product* of this former recognition stage. Contrarily, we have described a DAG generator more akin to the tradition of two-stage decoders. However, the novel inclusion of an optional feedback channel allows the use of LM information to improve pruning. Although both generation and DAG representation[23] are different, the graphs of this chapter and the time-conditioned word graphs of the aforementioned bibliographic references seem essentially equivalent. Likewise, the choice of using our feedback channel or the use of a unique time-dependent decoder stage seem also equivalent[24] *only* when constraining our dataflow components to use HMMs and regular LMs.

11.1.6 Clustering of frame boundaries

Word boundaries are not always precisely defined in ASR tasks and DAG edges between vertices varying only in a few frame boundaries are usually quite similar. This observation suggests an heuristic approach to reduce not only the size of the word DAG but also the computational cost required to generate it and, perhaps more importantly, to process it afterwards.

Although the idea of reducing the size of a word graph is not novel, we have to remark that most works on word graph minimization or compression (e.g. [Johnson and Harper 1999a; Weng *et al.* 1998]) are based on applying (sometimes heuristical) merging techniques over an already constructed word graph (which is, invariably the result of a previous decoding stage). These graphs represent the set of results of a decoder, including the contribution of the LM, and are not similar to the DAGs with generative likelihoods we are dealing here. As previously stated, our DAGs are more related to what some authors have called time-conditioned word graphs [Ortmanns *et al.* 1999]. Another nomenclature we have found in the literature is the distinction between LM conditioned and unconditioned word graphs[25] [Sixtus

LM conditioned vs unconditioned

22 Namely, labeled with generative likelihoods without the LM contribution.

23 Our DAG generation and decoding approaches do not need an explicit representation of the DAGs, they are serialized and, thus, processed while being generated, we are not aware of any other work sharing this feature.

24 In the sense that they compute the same outputs. There exists differences from other points of view as, for instance, the fact that the dataflow components can be reused in more orthogonal ways.

25 Despite this dichotomy, there exist, nevertheless, other types of graphs, as discussed in Section 5.6.

2003; Section 8.2]. Indeed, according to Sixtus, word graphs produced in a time-conditioned search are usually more redundant due to the presence of sentence hypotheses which only differ, slightly, in word boundary positions. A word boundary optimization step is usually performed to reduce this effect. No details are given relating this optimization step: the construction of time-conditioned word graphs is described in [Ney *et al.* 1997; Section 4], but nothing related to clustering vertices associated to similar time positions can be found. After some bibliographic search, the only explanation related to clustering word boundaries that we have found[26] is given in [Ortmanns 1998; Section 4.3.3.3] where the word graph is constructed from a tree of word hypotheses and vertices representing time instants are clustered when they are closer than a given threshold.

The idea described here is quite similar, but it is performed *on the fly* (while the DAG is being constructed). It consists in emitting a sole DAG vertex associated to a group of consecutive frame boundaries, as illustrated in Figure 163. More precisely, this must be detailed or distinguished for each state of the the control automaton, but let us ignore this for the moment in order to simplify the exposition. Instead of emitting DAG vertices at each frame boundary, the emission is delayed until the set of vertices which are to be clustered is closed. When a cluster of frame boundaries is created (by including its first frame boundary), a new lexicon decoder is constructed. This decoder is updated each time a frame is received, as usual. A difference w.r.t. the previous case is that new initial hypothesis are included at the beginning of the decoder at each frame boundary, as with re-entrant decoders. This produces the same output as creating a different decoder for each frame boundary in order to retain the best word hypothesis afterwards, but in a much more efficient way.

delayed emission

More and more frame boundaries are included in the group until one of the following conditions become true:

- the number of frame boundaries included in the group achieves an upper bound. This threshold is given as a parameter;
- the lexicon decoder associated to the group has an outgoing word hypothesis. Observe that ignoring this condition would imply creating a loop in the new vertex and the DAG property would no longer be preserved.

Once a new vertex is emitted, the next frame boundary starts a new group and a new decoder is associated to it. This approach produces DAGs with a number of vertices smaller than the number of frame boundaries. These vertices represent intervals rather than positions. The extension of this idea to take the control automaton into account is quite simple: one the one side, a different state is emitted for each active automaton state while the first component of the tuple ⟨frame frontier, automaton state⟩ becomes ⟨frame frontier group, automaton state⟩. On the other side, it suffices that at least one decoder of the pool generates an output hypothesis to force the entire group to be closed, no matter the number of automaton states affected by this decoder.

26 *In german!!* (a language I do not understand). We have been able to grasp the meaning thanks to an automatic translation tool.

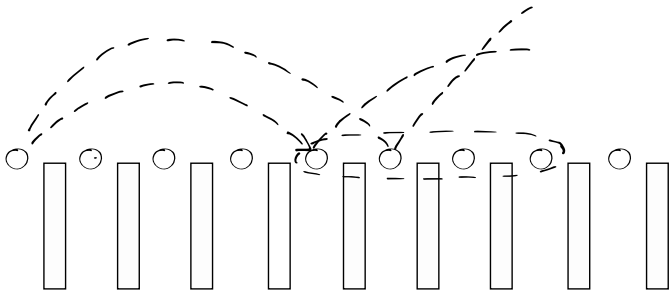


Figure 163: Clustering of frame boundaries: several consecutive frame boundaries are associated to the same DAG vertex (depicted as a dashed ellipse). Lexicon decoders are associated to these groups and created when the first frame boundary is included in the group. New hypotheses are created at the root of these decoders and the group is closed when the decoder is to emit their first outgoing hypothesis (or when then number of boundaries included in the group reaches a predetermined threshold).

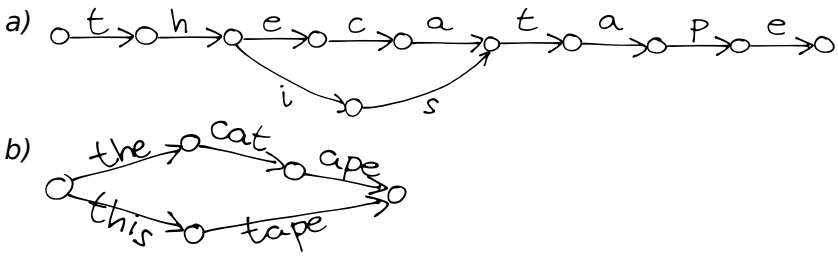
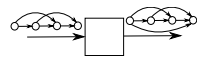


Figure 164: (a) a DAG labeled with sub-word units, and (b) a word DAG obtained from the previous one. Observe that no LM is required in this process so that, for instance, the score associated to word "ape" is independent of the fact that previous word was "cat" and so on.

11.2 FROM ANOTHER GRAPH (graph_graph_gen)

Previous section described a dataflow component to generate a DAG from a frame sequence. Another way to construct a DAG labeled with word hypotheses is to depart from a DAG of sub-word units, as depicted in Figure 164. The dataflow fulfilling this interface, described in Section 8.3.2, was called `graph_graph_gen`. It is a filter where both input and output DAGs are serialized with the incidence protocol.



The `graph_graph_gen` module is also based on a pool of lexicon decoders which use the transcription of words in terms of their corresponding sub-word units, as described in Section 10.9.5. However, these decoders would not perform segment estimation since this procedure has been previously applied to obtain the incoming DAG by means of a `seq_graph_gen`, as depicted in Figure 106.

Relating the exposition of this section, observe that the previous `seq_graph_gen` was first described in the most basic version in order to include, one at a time, more and more features. This approach was chosen mostly for pedagogical purposes. Now, since this section can

be viewed as a generalization of the former `seq_graph_gen` from input sequences to input DAGs, it is reasonable to describe the novel `graph_graph_gen` dataflow component directly with pruning and using a control automaton. The use of over-segmentation marks is briefly detailed since the extension of fat-sure frontiers is not straightforward. Besides, there are novel features which deserve dedicated subsections: the use of error correction edition operations and the capability to transmit the received feedback information back to the input.

As with the `seq_graph_gen`, we can observe in Figure 164 that, *in principle*, the set of vertices of the output DAG is a (not necessarily proper) subset of the vertices of the input DAG. Nevertheless, this observation does not necessarily hold due to the use of non-trivial control automata and, now, due to the optional edition operations. Relating word edges, they correspond to the sum[27] of paths in the sub-word DAG.

persistent estimators

Another already mentioned similitude with the `seq_graph_gen` component is the maintenance of a pool of decoders. However, there is an important difference (previously explained in Section 10.2.2) when switching from sequences to more general DAGs: we need to use persistent segment estimators since now a given decoder state can be used several times and, conversely, their state may come from several incoming or previous configurations, as illustrated in Figure 123. We can clearly observe in that figure that it is convenient to associate a decoder instance to each vertex. Let us also remark that the computation expressed in that figure should be understood as the combination of the entire input DAG with a finite state model. Assuming that the input DAG is labeled with sub-word units, this FSM could be the composition of the lexicon with the LM or, directly, the application of a sub-word based (a.k.a. lexicon free) LM. Note that is not what the `graph_graph_gen` is meant to do: only the construction of a word DAG from a sub-word one.[28]

As with the pool of the previous `seq_graph_gen`, new lexicon decoders are only started at some particular input DAG vertices: at the initial vertex and at each vertex where at least some word had previously finished. This behavior can be explained as the composition of the input DAG with the one-state automaton of Figure 160 described in Section 11.1.5. As with the `seq_graph_gen`, there is no point in having more than one instance of the same lexicon decoder type starting the estimation of words from the same input DAG vertex. This is the reason why we can associate, to each of these instances, the set of lexicon automaton transitions making use of this instance.

incoming tokens during main loop

Since the input DAG is received using the incidence serialization protocol, we can assume that the main loop of the algorithm would receive a `vertex` token indicating the presence of a new input vertex, followed by `edge` messages, and finished by a `no_more_in_edges` token. Each edge message contains the identifier of the origin vertex and a list

27 There can be several paths from one vertex to another describing the same word. The sum is in the semiring sense and is usually the maximum since we are assuming the Viterbi semiring.

28 Since the output of the `graph_graph_gen` is used by a DAG decoder afterwards, all these possibilities are, indeed, just a way to amuse or play with the associativity of transducer composition. Creating an ad hoc module such as `graph_graph_gen` may make sense in a dynamic decoding setting, but is not required in the static composition paradigm.

of tuples comprising, each one, a sub-word identifier together with its score. The destination vertex is not specified since it is necessarily the last received vertex. The `no_more_in_edges` message is used to notify the fact that no more incoming edges are to be expected.

pool representation

The data structure required to compute the generative likelihoods used to label outgoing edges maintains, associated to each previously received input vertex, a pool or set of persistent descriptions of lexicon decoders. These descriptions are removed or released when receiving `no_more_out_edges` messages.[29] More concretely, this set can contain, for each possible lexicon decoder type, as many decoders as predecessor vertices, each one related to the starting position where an initial decoder was first created.[30] However, only some of these possibilities are activated by the algorithm since a new decoder is only started when a state of the control automaton is activated and has an arc labeled with this decoder type, and this only happens at the initial state or when a decoder emits some output. The following explanation complements and tries to clarify the pseudo-code described in Algorithm 18.

- when a `vertex` message is received, it becomes the *current vertex* and an empty pool is associated to it. We have to check if the `vertex` message contains the `is_initial` flag. If so, the initial state of the control automaton is activated. Likewise, we check the `is_final` flag, but we simply store it to be used later;
- when an `edge` message is received, decoders from the origin vertex of the edge are traversed and the **`edge-step`** operation[31] is applied to them. This operation receives the list of ⟨word-id,score⟩ contained in the `edge` message. The resulting persistent decoder descriptions are stored in the pool when the pool does not contain them. In other case, they are merged with the previous decoder description contained in the pool by means of a **`merge`** operation;
- after updating the decoders of the pool due to the edge operations, the **`vertex-step`** procedure is applied to each decoder instance (as illustrated in Figure 123). Observe that this procedure is specially relevant when dealing with error correction edition operations (described below). At this point, the pool will no longer change and it can be considered *frozen*.

  This is the moment to ask decoders for outgoing hypotheses if they are required (related with the "generate output" and "generate marked" associated to the use of over-segmentation marks and described below). Since each decoder instance is associated

29 These messages are emitted, from time to time, due to pruning, by the DAG decoder (which is usually connected to the output of the `graph_graph_gen`). Each of these messages contains an input vertex identifier and the pool associated to this vertex is removed.

30 Otherwise stated, we consider, at each input vertex position, the estimation of edges starting at several previous positions. When these edges are computed by means of decoders based on regular models, we can incrementally compute a novel trellis column used not only to estimate the likelihood of words finishing at this point but also to pursue the estimation of word prefixes needed by the estimation of future output edges.

31 See Section 10.2.2 for an explanation of these operations including the **`merge`** operation described below.

to a list of control automaton arcs, we use these hypotheses to generate output edge messages between the corresponding output DAG vertices described by tuples[32] ⟨input DAG vertex, automaton state⟩ where the input DAG vertex of the origin is given by the pool and the input DAG vertex of the destination is always the current vertex. Hypotheses are also used to determine which control automaton states are active. Note that, even if there were no decoders or if their set of outgoing hypotheses was empty, the initial state is activated when processing the initial vertex. Observe also that these states can activate, in turn, other states due to null transitions (that are also emitted at the output DAG).

The possibility of emitting null transitions forces the DAG output vertices to be emitted in a proper order that can be pre-computed beforehand by computing a topological order of the control automaton w.r.t. null transitions.

Some data structures are used during these operations to prepare the set of outgoing vertices (and their respective edges) associated to the current input vertex.

The emitted vertices are marked as initial or as final if both input vertex and control automaton state are marked in the same way;

- the `no_more_out_edges` message can be used to release the persistent decoders associated to the vertex that will no longer be the origin of outgoing edges.

Observe that the pool associated to the current vertex is the only one that can be modified (with the exception of a complete removal due to pruning). It is also possible to delay the operations associated to the `edge` message until all these `edge` messages have been received. In that case, we simply store the messages and process them when the `no_more_in_edges` message is received. The advantage of delaying is that some slight optimizations can be envisaged.

11.2.1 Over-segmenter information

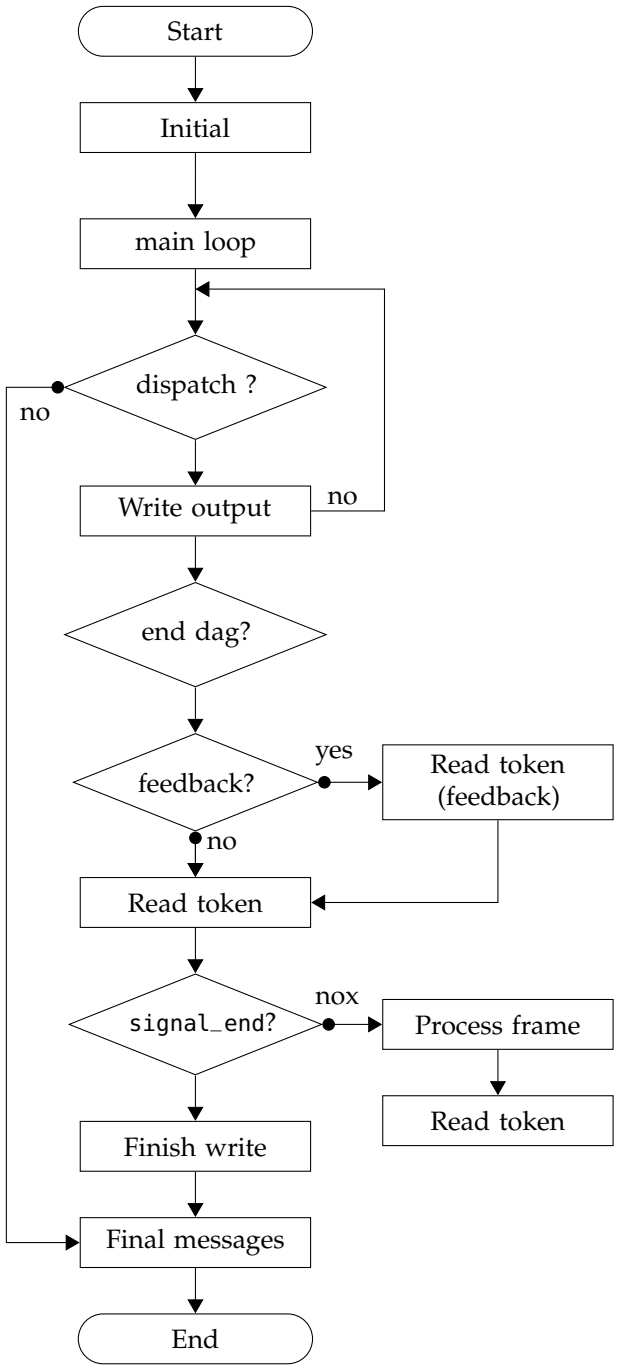
Frame boundary tokens were responsible for carrying the optional[33] over-segmentation marks in the previous `seq_graph_gen` dataflow component. Now, the obvious counterpart of frame boundary tokens in the input DAG are the `vertex` messages, which seem suitable for this task. It is very easy to deal with the following marks:

- for "non-sure frontiers" we proceed as defined before;
- the "sure no-frontier" mark is used to skip the generation of output hypotheses in lexicon decoders, at the current vertex, when processing the `no_more_in_edges` input token. As a consequence, no new hypotheses are started at this point;[34]

32 Converted into a numerical identifier (when emitting output tokens) by means of a dictionary.

33 Where, by "optional" we mean that the *by default* value was the "non-sure frontier" mark.

34 An exception being the case of the initial vertex. Although a sure no-frontier makes nonsense at the initial vertex, we can also choose to avoid the emission of words. This, obviously, would lead to an empty output DAG.



Algorithm 18: Flowchart of `graph_graph_gen`.

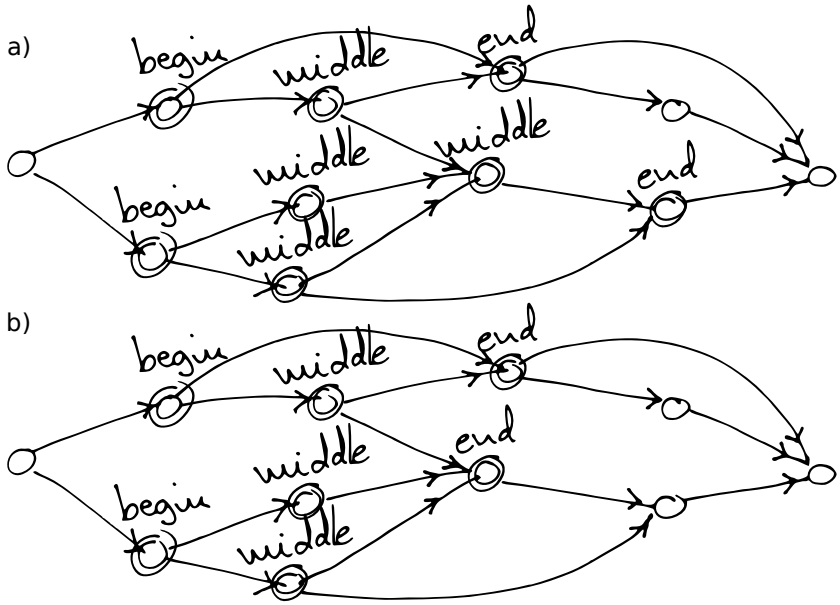


Figure 165: Fat sure frontiers in DAGs: (a) example of a DAG with a correct assignment of fat sure frontier marks, and (b) the same DAG with an incorrect set of marks. It is incorrect because of the path of the bottom where there is an edge from a "middle" vertex to outside of the fat sure group.

- the "sure frontier" means that not only the lexicon decoders are queried for outgoing word hypotheses, but also that only the decoders starting at this point will be propagated by the edges outgoing this vertex: Decoders others than those started at the current vertex are removed from the pool after querying their set of outgoing hypotheses.[35]

However, the case of a group of vertices jointly working as a sure frontier (what was known in the `seq_graph_gen` as a "fat sure frontier") is not so easy to generalize. These marks were associated, in the case of an input frame sequence, to a contiguous sub-sequence. An extension of this idea from sequences to more general DAGs can be as follows: we have to specify a set of DAG input vertices as beginning positions of the fat sure frontier, another subset as final positions and, finally, all the vertices between them as being "in the middle" of the group. The `graph_graph_gen` will assume that these marks satisfy the following property: "Any path crossing any of these marks would necessarily cross, in that order: an initial, zero or more middle and a final mark."

Since there can exist incorrect configurations, as depicted in Figure 165, it is important to warn that this property is not checked by our dataflow module and the `graph_graph_gen` would not work as expected without complaining about that.

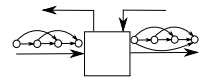
As with the `seq_graph_gen` dataflow module, we can factorize these marks in a set of flags identical to the ones already used in that component and shown in Table 7. These flags were used to put and remove

35 This turns to be a particular case of a very compact "fat sure frontier" described below.

a special "mark" in the lexicon decoders of the pool. Now, this feature has to be adapted to the case of persistent lexicon decoders since, in this case, there are persistent representations associated to a vertex and their "marks" have to be explicitly transferred to the new representations. Fortunately, we can observe that when some decoders are merged they always share the same "mark" flag value (i.e. only marked or non-marked, but never a mixture) so that it suffices to make the resulting decoders to inherit this feature.

11.2.2 Feedback channel

In typical recognizer examples making use of the `graph_graph_gen` component (see Figure 106) the DAG input is produced by a former `seq_graph_gen` which, in other circumstances, would send its output to a DAG decoder. This decoder is responsible for sending feedback messages back to the `seq_graph_gen`. Now, when using the usual combination of `graph_graph_gen` preceded by a `seq_graph_gen`, the DAG decoder sends the feedback to its immediate prececessor. This means that the `graph_graph_gen` is both a receiver and an emitter of feedback messages.



The feedback messages received by the `graph_graph_gen` are, each one, associated to a different output vertex. They contain the score of the best hypothesis, associated to this vertex, in trellis of the DAG decoder. As with the `seq_graph_gen`, this score is used to initialize the lexicon decoders of the pool that start, for its first time, at this point.[36] This score is meant to have the same pruning criteria in all lexicon decoder instances associated to the same input vertex. It will be properly removed when generating output word hypotheses. Since the use of a feedback channel is optional, the `graph_graph_gen` can perfectly work without it and, in that case, the best score of previous decoders from the pool is used instead.[37]

Observe that lexicon decoders stored in the pool are associated to the input vertices and not to the output ones (which where tuples ⟨input DAG vertex, automaton state⟩), so that scores obtained from the feedback channel have to be grouped by the corresponding input vertex value in order to obtain the maximal value from them.

Relating the feedback information sent back to the `seq_graph_gen`, we have to observe that, in this case, the `seq_graph_gen` is emitting sub-word units, making the effect of feedback, a priori, less relevant. Indeed, when this channel is not connected, the feedback mechanism can be disabled without problem. In other case, the proper way to compute the feedback message sent by the `graph_graph_gen` is to ask the lexicon decoders of the current vertex pool for their best score[38] and to send it back at this moment. This should be done when the pool has been frozen (after processing the incoming `no_more_in_edges` message).

36 More precisely, at the corresponding input vertex.

37 When the pool was previously empty, as is the case of the initial vertex, the by default initial mass (value one represented as zero in log-scale) is used.

38 Remember that these decoders do not estimate segments but only transform sub-word unit sequences into words. Note also that the best score is usually obtained in Viterbi decoders for pruning purposes.

11.2.3 Edition operations

An interesting generalization that will be considered is the presence of edition operations. Although these operations were first addressed in Section 6.9.1, our current discussion is more related to Sections 10.2.2, 10.9.5 and 10.4, where their use in decoders, when processing an input DAG, was the example chosen for motivating and describing the segment and lexicon estimator interface.

What does considering these operations in the `graph_graph_gen` dataflow module imply?

SUBSTITUTIONS only affect the edge estimators;

DELETION of edges means that edge estimators can not to consider a edge step but to decrease the computed probabilities by the edge deletion probability;

INSERTIONS means that we can emit new words without even considering the input. Without pruning, the output DAG could be infinite after the begin dag token. Even with pruning, the set of output DAG vertices is no longer the same as that of the input DAG.

To summarize, it does not suffice to use lexicon decoders taking edition operations into account. Its use is not completely transparent for the `graph_graph_gen` module due to word insertions.

Although insertions are constrained to the **`vertex-step`** operation, the entire `graph_graph_gen` has to be adapted to include an arbitrary number of output DAG vertices associated to them. We will also assume that lexicon decoders used internally by the `graph_graph_gen` are tree lexicons capable of dealing with insertions until their leaves are reached. This means that *they cannot emit word sequences by themselves*, but only complete the current word hypotheses. In this way, the `graph_graph_gen` should be modified to emit output DAG vertices after this partial **`vertex-step`** and enter in a loop where new output DAG vertices are created and connected with new edges only due to word insertions. This procedure should finish when pools become empty due to pruning,[39] although a maximal number of consecutive word insertions could be given as a parameter. In order to simplify the resulting model, a series of null edge transitions can be included to connect these intermediate output DAGs to the final one, as depicted in Figure 166 a). Let us remark that the use of a control automaton is reflected in the transitions between output DAG vertices and these transitions can also be observed in the vertices and edges associated to word insertions. In this case, the null transitions depicted in Figure 166 a) must be generalized as shown in Figures 166 b) and c).

Note also that the probability of inserting words departing from an newly created lexicon decoder is used many times and, since this is independent from the input DAG, can be pre-computed beforehand in order to be reused when needed.

39 A reason why the feedback mechanism should work inside this loop. Indeed, the DAG decoder using this output DAG would not be able to distinguish these word edges from the conventional ones associated to sub-word models first generated by a previous `seq_graph_gen`.

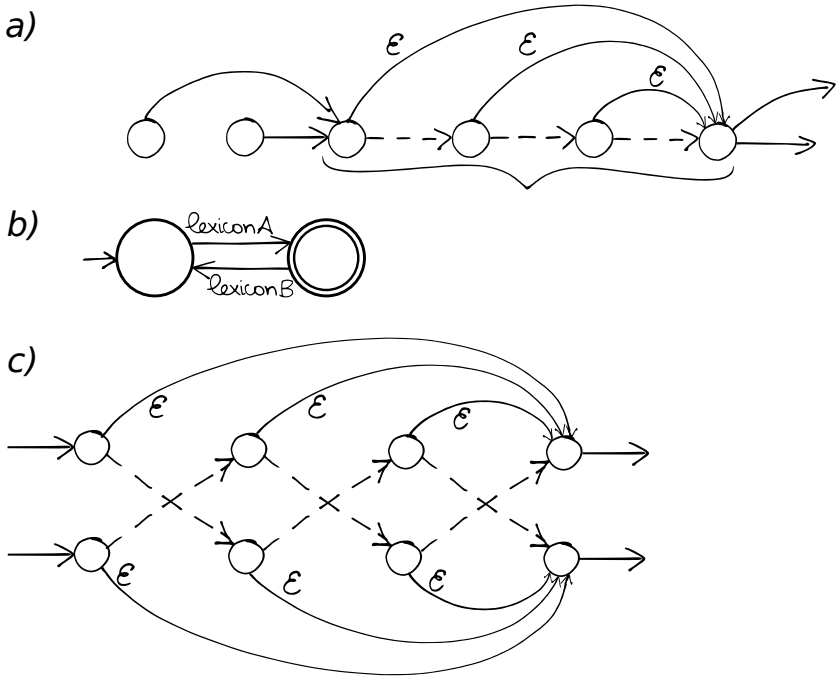


Figure 166: Dealing with insertions in the `graph_graph_gen` dataflow component: *a)* the set of DAG vertices grouped in a curly brace represent the equivalent of a DAG vertex when the insertion edition operation is not taken into account. The number of states depends on the number of consecutive insertions before the pruning is high enough to stop the loop. Edges depicted with dashed lines are generated by insertions, but they cannot distinguished from other edges by the component consuming this DAG. Null transitions simplifies the treatment of these operations since they make the insertions optional, *b)* a control automaton used in the example depicted in *c)*, which shows that the control automaton transitions must be taken into account when processing insertions.

Other possible extensions

Let us finish the section devoted to the `graph_graph_gen` component by remarking that this module has been further extended by Pako Zamora, a member of our research group, in his PhD work [Zamora-Martínez 2012] to deal with reordering in order to work with SMTs tasks. Besides reordering, a remarkable feature of his extension is the capability to work efficiently with multi-stage trellises (briefly described in Section 8.2).

Although the core features described in this work are due to myself, there is a natural overlap with [Zamora-Martínez 2012] since the implementation of the dataflow components, as well as the majority of April toolkit,[40] was a joint collaboration between myself, Pako Zamora and Jorge Gorbe.

40 As explained elsewhere, April toolkit has now a major fork called April-ANN and mainly due to Pako Zamora and related to ANNs. It is expected to merge the work of this work (dataflow architecture and decoders) in a near future.

11.3 COMBINING DAGS (FOR MULTI-MODAL AND MULTI-STREAM DECODING)

The combination of different information sources is a common technique in pattern recognition in general and in tasks more related to our work (e.g. ASR or HTR) in particular. Multi-stream processing and multi-modal recognition, briefly reviewed in Section 3.2.1, constitute two different examples of this combination. We saw in that section that early fusion and late fusion techniques could be used for this purpose. This is also related to the use of ensembles where the output of different decoders applied to the same input is combined. The use of different not too correlated recognition systems usually leads to improved results.

Since some combination techniques can use the set of best hypotheses of each system (instead of being restricted to the use of the 1-best hypotheses), the combination of word graphs naturally arises due to the fact that such graphs constitute a compact way to encode these sets. Although this combination is usually performed on DAGs containing posteriors, we are now interested in combining graphs scored with generative likelihoods, as those produced by the dataflow modules described in previous sections. This combination can be used by a DAG decoder afterwards, as exemplified in the recognition architecture described in Section 8.4.6 and illustrated in Figure 112.

The dataflow component required for this purpose takes as input several DAGs and produces, at the output, a new DAG which can be seen as the combination of the input ones. Although we will limit our study to the combination of just two inputs, the algorithm can be easily generalized to more inputs.

11.4 SUMMARY AND SOME CONCLUSIONS

This chapter has addressed the algorithmic and implementation details of the first stage of two stage decoders described in Chapter 4. In this setting, a DAG is usually proposed as a compact way to represent the set of segments of the input signal. The classical point of view assumes that vertices represent input positions (e.g. time instants in ASR) whereas edges, representing segments, may be labeled with sub-word or word identifiers together with their generative likelihoods. However, this property no longer holds when abandoning the simplest independence assumptions relating the generation of segments. Although these assumptions were organized in a hierarchy in Section 4.2.1, we have only taken into account, for the DAG generation of this chapter, the use of context dependent units, disregarding other features (related to the segment neighborhood) that would lead to much more complex DAGs. In this way, a DAG vertex is no longer associated univocally to an input position since several vertices corresponding to the same position may have to be distinguished by their phonetic[41] context.

41 Phonetic in ASR, although context dependent units can be applied in other tasks.

Although the DAG generation procedures described here could have been implemented in other ways, we have circumscribed them to the dataflow architecture of Chapter 8 in order to be used as basic components of the recognizer examples of Section 8.4. This has lead to a reactive programming style. Besides, the proposed dataflow components will process their input from left to right with top-down filtering and making use of the DAG serialization protocol that we have designed with both DAG generation and DAG decoding (addressed in Section 12.2) in mind.

The constraint to proceed in an online way with the serialization protocol has the advantage of allowing one module to process a DAG while it is being generated without the need to store or to explicitly represent it in the computer memory. Besides, our implementation is based on a finite state skeleton (avoiding the simpler use of, for instance, mutexes and threads and explaining why some algorithms have been reported as dataflow charts) which leads to a much lower overhead. The drawback of this approach is that the implementation is probably more cumbersome than otherwise.

Several DAG generators have been proposed: the first one, that we have coined `seq_graph_gen`, is directly related with the two stage of the TSGM mentioned so far. The second component, more general, obtains a DAG from another one. They only differ in the input modality and the type of lexicon decoder used in their respective pools. They are highly related since the first one can be seen as a common special version of the second. The rationale for this specialization is twofold: 1) the use of more efficient ephemeral or in-place updating versions of lexicon decoders, and 2) a simpler input protocol based on a frame sequence instead of using the DAG serialization protocol.[42] A third generator module performs a combination of several input DAGs and is intended for multi-stream and multi-modal recognizers when we know that different inputs corresponds to the same word sequence. Instead of performing a different decoding on each input stream in order to combine the results afterwards, we can use the proposed DAG combination module to obtain a unique input for a sole DAG decoder.

If we can usually found similarities between the proposed techniques and the state of the art, it is more difficult to discover, in the vast literature, ideas equivalent to the ones proposed in this chapter. Although the kind of decoders based on these modules are sometimes similar to the time-conditioned search [Ortmanns and Ney 2000], it would be unfair to limit our comparison to them since, in principle, our algorithms are able to deal with segment models other than HMMs and, as shown in Chapter 8 (concretely, in Section 8.4), it is possible to design a large plethora of recognition architectures just by combining a few basic pieces metaphorically emulating "lego pieces" that can even be distributed among different computers. Indeed, we have not found any decoder making use of a dataflow architecture in the same way.

Another difference with related graph generation techniques, besides the mentioned capability of using a wider range of segment model types, is the use of over-segmentation information. A new type

42 Although this simplifies the parametrization and over-segmenter modules which feed the `seq_graph_gen`, there is a direct correspondence between the frame and frame-boundary tokens and messages that could receive the `graph_graph_gen`.

of over-segmentation described here is novel (to the best of our knowledge) and is motivated by our experimental work in HTR, described in Chapters 14 and 15. Another novel feature is a feedback mechanism allowing the DAG decoder to improve the pruning techniques to the extent that the semi-decoupled architectures proposed in this work (see Section 8.4), when including this feature, are roughly equivalent to an integrated decoder while being constructed in a much more modular way.

The use of a control automaton seems a proper way to model different features (e.g. across-word context dependent units, or the inclusion of optional silences[43] that are transparent to the subsequent LM). Indeed, it is just a particular case of the widely known transducer composition paradigm. Even if this feature seems straightforward from the point of view of the static network composition approach, their inclusion in our graph generation modules has some particularities which deserve to be explained. Indeed, this peculiarity has no counterpart in the most common implementation of the dynamic approach (indeed, their flagship): the one-pass decoder with a re-entrant tree lexicon using the token passing paradigm.

Some readers may also find a slight resemblance between the semi-decoupled architectures of this work, the layered approach of [Demuynck *et al.* 2003], the bottom-up proposal from [Siniscalchi *et al.* 2013] or even the phone-to-word decoding described in [Bertoldi *et al.* 2008]. They mainly share with our systems the fact of using a sub-word DAG representation at some intermediate stage: in [Demuynck *et al.* 2003], a phone network is generated in a first stage in order to be decoded in a subsequent one. The second stage makes use of error correction edition operations during decoding as with the proposed `graph_graph_gen` extension of Section 11.2.3. The approach presented in [Siniscalchi *et al.* 2013] (see [Siniscalchi *et al.* 2013; Figure 4]) puts the emphasis on a decoupled, bottom-up procedure based on the generation of a phone DAG that is used to generate a word DAG in order to, finally, apply the LM to it. Excepting lots of details,[44] the most similar thing described in our work is the use of a `seq_graph_gen` to produce sub-word based DAG that is converted into a word DAG by the `graph_graph_gen` in order to send it to the DAG decoder, as illustrated in Figure 106. Finally, the phone DAGs used in [Bertoldi *et al.* 2008] are transformed into confusion networks prior to their decoding. Besides, the first phone graph is obtained by a prior decoding stage based on a lexicon-free or sub-word based LM. As can be observed, all these alternatives are quite different from our proposals.

43 Including inter-word, initial and final specialized models and in such a way that chained silence edges are forbidden.

44 The emphasis of the work is on the application of a bank of parallel, specialized attribute detectors for ASR related features (e.g. nasal detector) used to obtain phone posteriors at the frame level. Although the origin of this works seems to be the desire to overcome the limitations of the beads-on-a-string paradigm, this seems very related, in our opinion, to HMM based on the estimation of frame posteriors as described in Section 7.5.3.

12 DECODING

What is the probability that Shakespeare,
by randomly flexing his muscles,
might accidentally have found himself
swinging through the trees like a monkey?

John Allen Paulos

CONTENTS

12.1 INTRODUCTION

why this chapter is so brief

Although this chapter is devoted to both one-pass and two-stage decoders, it is very brief. The reason is that several parts of these decoders have been described elsewhere: For the two-stage decoder we will only address its second stage, since the first one was the subject of Chapter 11. We will see that this second stage is basically a dataflow component to combine a DAG (received by means of the serialization protocol) with a LM (using the LM interface described in Section 9.1). Relating the one-pass decoder, the most relevant parts required for its construction have been reported in Chapters 10 and 9. We have already seen[1] that there are mainly two different approaches to design one-pass decoders: dynamic and static. We have proposed a dynamic decoder which makes use of the lexicon decoders and LM estimators of previous chapters, taking advantage of some particular features of these sub-components to maintain a representation of active states in a cache friendly way that has the particularity of never requiring explicit searches (e.g. hashing) of active states. We show how this particularity, already present in some lexicon estimators of Chapter 10, can be extended to the overall one-pass recognizer. Other noteworthy feature is the use of bunch LM queries of Section 9.1.2.

1 In the introduction of Chapter 10.

12.1.1 Parallel, multi-threaded and distributed recognizers

Before entering into the substance, and since we will discuss how decoders can be parallelized, let us first make a very brief aside on parallelism, on distributed architectures and also on the efficient use of certain properties of most popular current computer architectures.

Nevertheless, it is first convenient to ask ourselves which purposes are to be achieved with these techniques. Many readers may seem this a too trivial question which does not deserve even mentioning it, but we will see that there exists several scenarios and we hope that some insight can be obtained from them. Some of these purposes are:

IMPROVE THROUGHPUT It is obvious that we do not need to care about parallelism in order to process millions of independent sentences: this is a clear example of an embarrassingly parallel problem. It suffices to distribute different sentences into independent recognizers. However, even in this case the problem of running several recognizers in the same multi-core machine to obtain the maximal throughput requires several considerations related to memory bandwidth, sharing resources and so on.

ACHIEVING AND MAINTAINING REAL-TIME seems mandatory even for the most modest devices. Is therefore possible to envisage a paradigm where decoders may tune their performance by increasing the pruning criteria or even by gracefully lowering the quality of their models when other user processes demand more resources, when the battery is low, etc.

PROCESSING EXTREMELY LONG SEQUENCES is usually addressed by means of explicit techniques based on not storing all the information required after dynamic programming for traceback purposes. Although we will not deep into details, we can mention the possibility of storing checkpoints [Bax 2005] or the use of partial traceback [Brown *et al.* 1982] which is often used for dictation tasks or when desiring to process the output while the input data is being received. Nevertheless, processing long sequences can also benefit from parallel and distributed environments for several reasons. Distributed systems not only distribute the huge load but usually means also more (distributed) memory.

Parallelism can be counterproductive in many cases. A too fined grained parallelism may be inadvisable. Also, even for embarrassingly parallel tasks, it may be convenient to consider distributed systems in order, for instance, to share resources. An example is the use of large LMs which require large amounts of memory, which explains the convenience to establish a dedicated LM server.

Another important feature related with current computer architectures is the memory hierarchy. Roughly speaking, this means that there are several memory types from registers and cache ram to secondary memories. The penalty for accessing the main memory explains why data locality is so important for practical purposes and also justifies the existence of cache aware and cache oblivious algorithms. These terms refer to algorithms that take into account the existence of a two-level cache so that their design try to reduce the number of cache

misses. While cache aware take into account the sizes of the cache and the cache pages, cache oblivious only requires to assume the presence of a cache. Most newly specialized decoders algorithms proposed in Chapter 10 are cache oblivious.

cache aware & cache oblivious

Even without necessarily resorting to this kind of algorithms, and as a rule of thumb, we can mention the convenience of processing the data in such a way that the tasks involving the same data and algorithms are jointly processed to reduce the memory bandwidth requirements between cache and main memory. An example is the capability of the LM interface described in Section 9.1 to take a bunch of queries into account and the properties of both count based representation (our `lira` model) and for NN LMs where the enhanced possibilities of parallelism, when several queries are available, are widely known. Another obvious example of the importance of taking profit of hardware particularities is the use of specialized vector instructions (SIMD extensions available in many modern CPUs) and even the use of GPUs for numerical computation (e.g. training and evaluating GMMs and connectionist models commonly used for estimating likelihoods and scaled posteriors, not to mention the use of these connectionist models in NN LMs). There are many more relevant questions such as the distribution of tasks in threads and the placement of these threads on the different cores of a multi-core system and the issues that may appear due to cache coherence, etc. All these examples clearly give support to the importance of the second item of the "successful trilogy" of [Aubert 2002] discussed in Section 1.2: *"a clever design and implementation cooperative with the hardware."*

12.2 TWO STAGE DECODER

Chapter 8 described a dataflow architecture and, in particular, some two-stage architectures where described in Sections 8.4.2 and 8.4.3. These decoders where based on running the two stage generative model of Chapter 4 in a reversed way. Decomposing the problem into two stages makes this approach very flexible: several features can be addressed in the first stage (across-word context dependent modeling, multi-modal input, etc.) without any need to modify the second stage.

The first dataflow component of this scheme was the subject of Chapter 11 and what remains to be explained is a decoder which receives the input DAG serialized with the incidence protocol of Section 8.2. This protocol allows the input DAG to be broken into little pieces which may be sent at the same time they are produced. But this is not the only advantage: this protocol is very suitable to express the decoding algorithm in a reactive programming style, that has also been used in other previous dataflow components, as *"the set of actions to be performed every time a message is received"*. Let us describe which data structures and objects are needed, how they are modified every time a DAG serialization protocol message is received and, finally, which messages must be sent and which values are the result of the computation. The basics of this computation correspond are the subject of Chapter 5.

reactive programming style

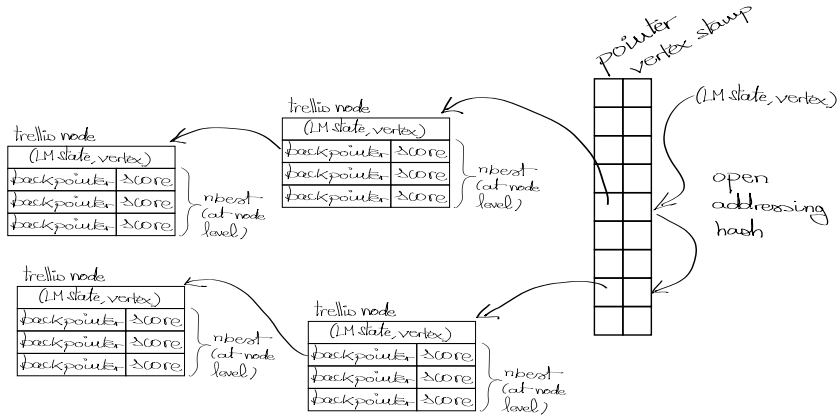


Figure 167: A sole open addressing hash table suffices to index trellis nodes since the nodes associated to the current vertex have to be indexed by their key ⟨input vertex, LM state⟩ . When the current vertex changes, the hash table is automatically cleaned and detached in constant time.

12.2.1 Internal components

The most important internal component is the trellis data structure, which can be returned as the final result of the decoding process or can be used to obtain the best sequence or other output modalities. This data structure contains a node for every tuple ⟨input vertex, LM state⟩ reached during the search. Every trellis node also contains backpointers to other trellis nodes.

Thanks to the incidence protocol, only the nodes associated to the current vertex have to be indexed by their ⟨input vertex, LM state⟩ value. Trellis nodes related to previous vertices are only required, during decoding, when they correspond to a vertex which has not yet received the `no_more_out_edges` message. Moreover, these previous vertices are traversed by using an iterator, so that they do not need to be indexed by their ⟨input vertex, LM state⟩ value.

We propose to use a hash table with open addressing collision resolution in order to index the trellis nodes associated to the current vertex. We only need to perform insertions and searches in this table, since deletions are not required. When no more incoming edges arrive at the current vertex, the entire hash table is detached from all nodes it was pointing to, so that it can be reused to index new trellis nodes which will be associated to the following vertex.

The proposed open addressing hash table is depicted in Figure 167, and contains nodes composed by the following fields:

- a pointer to the actual trellis node. In this way, it is not required to store the key into the table;
- the value of the current vertex, also called stamp[2].

2 The name "time-stamp" is more known and has been widely used in many ASR decoders, but input dag vertices are not related to time in many tasks. For instance, these vertices are related to horizontal coordinates in offline HTR. In general, DAG vertices

The stamp is an integer which is compared with the current vertex identifier in order to determine whether or not a given hash node is free. This feature allows cleaning the entire hash table in constant time since, when the current vertex changes, all the nodes become automatically free. Note that this value is initially set with a value which will never be used as vertex identifier. This technique was also used by algorithms described in Section 10.3.2.

The trellis data structure requires another table called "trellis vertex table" to obtain, for any input vertex, the set all its trellis nodes.

The proposed decoder also requires a reference to a LM object (exposing the LM interface described in Section 9.1) in order to obtain the initial and final LM states, and also to perform LM transitions.

12.2.2 Behavior

In order to build and run a dataflow graph it has to be constructed by first creating the required dataflow components and by connecting their output ports to the corresponding input ports afterwards. Our decoder dataflow component contains a sole input channel, where the DAG is received, and an optional output channel used to send feedback information to the DAG generator, as described in Section 11.1.3. The trellis data structure updated during decoding can be used to obtain the desired results. Although it is possible to serialize and to emit the trellis data structure in the form of dataflow tokens, we have opted, in the current implementation, to extract these values from outside the "dataflow world". Both 1-best and N-best solutions are available as well as the possibility of dumping part[3] of this trellis on a word graph. In order to obtain the N-Best solutions, we have implemented the Lazy Eppstein algorithm proposed in [Jimenez and Marzal 2003].

Let us now review the different types of message of the incidence serialization protocol (Section 8.2) may arrive by the input channel and which behavior is associated to each of them.

`begin_dag` When this message is received, the trellis data structure is guarantied to be empty.[4]

`vertex` It suffices to update the current vertex identifier. This attribute is used by the hash table to determine if a given hash node is empty or contains a pointer to a trellis node.

`is_initial` When this message[5] is received, a new trellis node, intended to be the initial node, is constructed and inserted into the

are related to boundaries between adjacent frames but, when using context dependent units leads, frame boundaries are no longer associated to a unique DAG vertex.

3 A pruning is applied which ranges from the removal of non-useful parts to select the edges used by the N-best algorithm. Some improvements are envisaged.

4 As described above, hash nodes contain a current-vertex-stamp different from all possible vertex identifiers.

5 We can assume, without loss of generality, that there is only one initial LM state, since null transitions can be used to emulate a LM with multiple initial states. Nevertheless, the most common LMs are n-grams which are deterministic and only contain one initial state. It is also possible to assume that there is only one initial DAG vertex, since the input DAG may also contain null edges. Remember also that, according to the incidence protocol, there are three tokens related to the initial vertex: the `is_initial` message is preceded by a `vertex` message and followed by a `no_more_in_edges` message (where LM null transitions would be processed, if necessary).

```
r ← iterator associated to origin vertex from the
    "trellis vertex table"
while r ≠ end iterator do
    s ← iterator associated to edge attributes
    while s ≠ end iterator do
        t ← iterator from LM.transition(r.LMstate, s.wordId)
        while t ≠ end iterator do
            /* this method creates a new trellis node when
               necessary */
            n ← trellis.createSearchNodeByKey(currentVertex,
                                              t.destLMstate)
            n.update(r,t.transProb * s.wordProb)
            t ← t.next
        s ← s.next
    r ← r.next
```

Algorithm 19: `edge` method in an input DAG decoder.

trellis. Vertex values are obtained as follows: the vertex identifier is obtained from the current vertex, the LM identifier associated to the LM initial state is obtained from the LM object associated to the decoder and, finally, the score corresponding to the initial (scaled) probability mass is obtained from the initial vertex score (an argument of the `is_initial` message received by the decoder) and from the LM initial state.

`edge` message, received for each input DAG edge, contains:

- the origin vertex;
- (the destination vertex is implicitly the current vertex); and
- the edge attribute is a list of ⟨symbol, score⟩ pairs.

These values are used to create or to update a possibly large set of trellis nodes associated to the current vertex, as shown in Algorithm 19: the set of trellis nodes created/updated and associated to the current vertex are obtained by taking each trellis node associated to the origin vertex of the edge and each one of the ⟨symbol, score⟩ tuples from the edge message. For each LM state and symbol, an updated LM state is obtained and is used to locate or to create the new trellis node associated to the current vertex. A pointer to the original trellis node (which become the previous one) is kept for the best scores associated to the current vertex and LM state pair. The inner loop allows the use of non deterministic or incomplete[6] LMs, although it can be specialized to the case of deterministic and complete LMs such as the common case of n-grams. This procedure can be optimized by first obtaining the best score associated to every word by considering all edges incoming the current vertex at the same time.

6 It is possible not to have transitions with a given word. If this seems unlikely due to smoothing, we have considered the possibility of pruning inside the LM.

```
r ← iterator associated to origin vertex from the
    "trellis vertex table"
while r ≠ end iterator do
    /* this method creates a new trellis node when
       necessary */
    n ← trellis.createSearchNodeByKey(currentVertex, r.LMstate)
    n.update(r,r.prob * nullEdgeTransition)
    r ← r.next
```

Algorithm 20: `null_edge` method in an input DAG decoder.

`null_edge` message is very similar to the previous message type,[7] but represents a DAG edge without symbols attached to it. In this case, a single score replaces the list of ⟨symbol, score⟩ pairs which can be found in the `edge` message. The method applied to update the trellis in this case (Algorithm 20) is a simplification of the previous one because LM transitions are not required.

`no_more_in_edges` message is sent to the decoder after all edges arriving to the current vertex have already been sent. The decoder may take profit this message to send a feedback message to the DAG generator. This feedback is intended to improve the DAG generation performance, as explained in Section 11.1.3. Two types of information have been envisaged:

- the best score of all active hypothesis associated to the current input DAG vertex;
- which symbols/words may follow, with a non negligible probability, from the set of active LM states associated to the current vertex.

When the decoder is not connected to the DAG generator by a feedback channel, the feedback message is not even computed. Our current implementation only sends the best score[8] which is an upper bound of the best use the hypotheses associated to the current DAG vertex (this bound contains information related to the LM, which is not present in the DAG generator). This value allows the DAG generator to use a unique pruning criterion for all hypotheses inside lexicon decoders no matter when (in which frame boundary) they have started. As explained elsewhere, this makes the proposed two stage decoder to behave nearly like an integrated system despite being made from two pieces implemented separately. Note that this feature is only related to the pruning of the search space.

The other type of feedback information proposed before is related to the minimal lexicon required by the set of active LM states associated to the current vertex. This might be useful to dynamically select the tiniest lexicon required by the LM. Note

7 Note that a special "null symbol" could have been processed by the edges message. We have decided to use a separated message because the code to deal with null transitions is different, but the other option can be easily implemented.

8 Which is usually also computed as part of the pruning process.

that this information can be sent very compactly: a sole numerical value indicating the lowest common ancestor of all lexicon required by every active LM state. We do not enter into more detail because, unfortunately, this feature seems not practical when using smoothed n-grams, as is usually the case: For these models, the entire lexicon is reachable from any LM state, leading always trivially to the entire lexicon decoder. We describe this possibility for completeness and because we have never seen this idea before. Maybe, semantically restricted domain tasks might benefit from this technique but, since we are not dealing with any of these tasks, we cannot provide experimental results.

release resources

no_more_out_edges message is sent by the DAG generator when all outgoing edges from a given vertex have been sent. This message allows the decoder to release resources associated to that vertex. In this case, we could take profit of this information to prune trellis nodes without outgoing edges, since they are no longer reachable.[9] Despite being easy to implement, we do not use this feature because we do not deal with very long sequences and it is simpler to remove the entire trellis once it has been used to extract the best solutions.

end_dag is used to track the final trellis nodes used later during the traceback of the best solution(s).

We can conclude by observing that this is just the reactive programming way to implement the FST composition algorithms of Chapter 5 (specialized for acyclic inputs and not limited to strings, as explained in Section 10.2.2) using the dataflow architecture and the DAG serialization protocol. Relating parallelization possibilities, the point where most improvements could be done is when processing the `edge` messages, specially when jointly processing all the edges incoming to a given vertex. In that case, the edge message would simply keep track of the information and the processing would be delayed to the `no_more_in_edges`, allowing not only to factorize the hypotheses associated to the same words but also the use of bunch LM queries.

12.3 ONE PASS DECODER

Although we have been interested in more general scenarios, most ASR and HTR systems are limited to inputs in the form of frame sequences, HMMs to model sub-word units and n-grams for language modeling. We know that the one pass decoder is, for this combination of features, much more suitable than the two stage: a dynamic programming algorithm computes, for every state of a huge HMM (constructed by expanding words in the LM using the lexicon and by expanding sub-words units using the sub-word HMMs), the score associated to the best way to reach this state at every input sequence frame boundary. The cost is linear with the number of frames.[10]

9 This operation should not be performed on the final vertex.

10 And also linear with the maximum number of active states, which are controlled by means of pruning techniques. But note that the size of the input DAG in a two stage

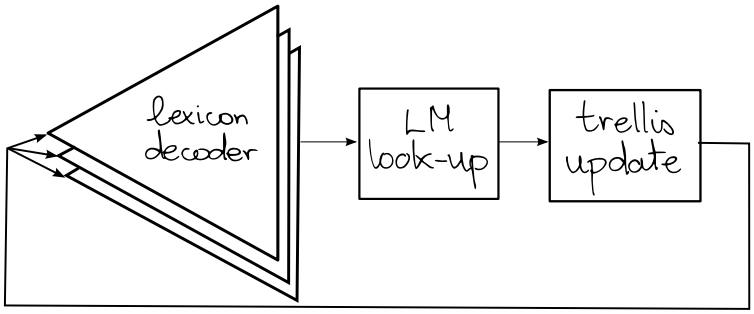


Figure 168: General scheme of a LM conditioned one pass decoder showing the three main parts: Several lexicon decoders are usually active, each one contains active hypotheses at the lexicon level sharing the same LM stage. Word hypotheses generated by lexicon decoders are sent to the LM updating module where LM state and hypothesis scores are modified accordingly. The trellis node pointers later updated to keep track of the previous history and sent back to their corresponding lexicon decoders, which may have to be created.

The approach followed in this work corresponds to the dynamic approach[11] in the form known as "LM conditioned one pass decoder", and uses the token passing paradigm described in Section 10.2.3: lexicon decoders as those proposed in Chapter 10 receive tokens and input frames. A general scheme of this type of decoders is shown in Figure 168, where three main parts can be observed:

token passing

- Lexicon decoders: Each one is associated to a different LM state and contains a reference to a (usually tree) lexicon model and a set of active states hypotheses. During a Viterbi decoding step, a frame, an over-segmentation mark associated to the following frame boundary, and a set of input tokens are used to produce a new set of active hypotheses and a set of output hypotheses. The set of hypotheses generated at the output of lexicon decoders contain, each one: a score, a LM state, the hypothesized word and a pointer to a trellis node. Tokens are propagated inside lexicon decoders, but only the score field are modified.
- A LM module updates the set of active hypotheses generated by the lexicon decoders: for every hypothesis it performs a LM transition from the LM state using the hypothesized word. The LM state is replaced and the LM transition probability is accumulated into the score. Note that non deterministic LMs may lead to zero, one or several destination states, so that the number of active hypotheses after the LM updating stage may be different.

decoder may grow quadratically with the number for frames: The number of segments on a two-stage decoder is related to the possible spans. This may grow quadratically unless an explicit or implicit (via pruning) bound on the length of longer segments is applied. In that case, the cost becomes linear, as explained in Chapter 11.

11 As explained before, there are roughly two broad families of one pass decoders: those using a static network representation and others known as "dynamic approach". Both types perform a similar computation, but the results may differ because of the different use of pruning. Note that the static network representation can be constructed on the fly. Every piece in the static network representation can be accessed in a uniform way with an automaton interface (Section 9.1.2) by on-the-fly determinization algorithms.

Note that this approach allows the use of back-off smoothing with n-grams since the word identity is known.[12]

- The trellis node pointer contained in every hypothesis is also updated using the original trellis node pointer, the frame boundary index (time in ASR) and the hypothesized word to obtain a new trellis node. It is possible therefore to retrieve the segmentation at the word level, but not at the sub-word level.[13]

The set of lexicon decoders contains at most one active hypothesis, and hence one token, for a given combination of lexicon network state and LM state. It is very common to maintain a separate copy of lexicon decoder for each LM state. A drawback of this approach is that a dictionary has to be used to locate the lexicon decoder associated to each hypothesis. When a LM state is not found in the dictionary, a new lexicon decoder copy has to be created. Conversely, when a lexicon decoder has no active hypotheses, it must be deleted and removed from the dictionary in order to save space.

12.3.1 One pass without dictionary look-ups

The use of a sole lexicon decoder for all hypotheses, no matter their LM state, overcomes some problems of tree copy decoders: the dictionary to map LM states to lexicon decoder copies, as well as the creation and removal of lexicon decoders, are no longer required.

Although the idea of maintaining several tokens, which do not compete among them, in the same state of a static lexicon decoder is not new,[14] some particularities (mainly described in Section 10.6.9) are novel: the idea of storing consecutively the tokens which share the same tree lexicon state, sorted by their LM state and using this order to simulate the merge procedure from merge-sort to easily detect collisions, is novel, to the best of our knowledge. We have combined this ordering with the (possibly reversed) topological ordering of tree lexicon states in order to store *all* lexicon hypotheses in a unique vector which is traversed once for reading after being created in a consecutive way. In this way, dictionary look-ups are not only completely avoided, but also the memory access pattern takes a better profit of the cache.

The proposed decoder is a direct application of the ideas described in Section 10.6.6, although additional implementation effort has been done here in order to paralellize things as possible by means of posix threads, mutexes and FIFO queues.

A one pass decoder dataflow component, as described in Section 8.4.1 and illustrated in Figure 104, receives the same type of input as the DAG generation of the two stage decoder (Section 11.1): a sequence which alternates frames and frame boundary information. This decoder would just modify a trellis data structure just as the second stage decoder described in the previous section. However, it is also possible

12 Refuting the remark found in [Llorens Piñana 2000; Page 50] and making unnecessary the use of finite state automata with null transitions approximations for these models.

13 Since we are not interested in this lower level of detail, we save a lot of spatial and temporal resources.

14 We can find some resemblances with ideas exposed in [Alleva *et al.* 1996; Finke *et al.* 1999; Huijbregts 2008].

to include the preprocessing, feature extraction and likelihood estimation in the same thread based scheme and the dataflow architecture would not be required.[15] We have currently implemented a version of the one-pass decoder without the dataflow architecture for speech dictation[16] and the following threads have been used to take profit of multi-core architectures:

- The lexicon decoder runs in a thread. This decoder uses a threshold value to skip hypotheses from the previous set of active hypotheses and produces a new vector of active states. The threshold value is computed from the entire set of lexicon hypotheses using a minheap of scores whose size is the maximum number of hypotheses. This has the same effect as histogram pruning.
- The beam threshold used by the lexicon decoder in the next decoding step is computed in a separate thread. This thread reads the vector of hypotheses the lexicon decoder is writing, so we need to solve the typical producer-consumer pattern: The consumer thread processes the vector up to a given size which is read from a mutexed variable written by the producer (the decoder). The lexicon decoder updates this value every N hypotheses[17] in order to reduce synchronization overhead.
- There is another thread to perform the LM trellis updating steps. This thread receives the set of active hypotheses generated by the lexicon decoder and returns, after some time, the updated version. It is very easy to process these LM queries in bunch mode (Section 9.1.2).
- Note that the computation of HMM emissions from the frame sequence can be computed in a different thread no matter the entire decoder is serial or parallel. We usually use HMM/MLPs and the computation of emissions by MLP is performed in bunch mode. The use of GMMs with context dependent units makes more questionable this a priori computation of HMM emissions since only some of them could be used and an on demand computation would reduce computational cost but, with MLPs, all emissions are computed at the same time.

The tree lexicon decoder used by this decoder processes active hypotheses from leaves to the root, and we have taken profit of this fact with special care in order to overlap as much as possible the different tasks,[18] as explained in Figure 169. As soon as the lexicon decoder has generated the set of output hypotheses, it can immediately start the following decoding step, provided the following frame HMM emissions are available. It does not need to wait for the results computed

15 One of the advantages of the dataflow architecture is the serialization protocol that is not used in this particular case.

16 It is based on a partial traceback procedure to emit words as soon as the currently live hypotheses share a unique prefix. This ASR system, based on frequency filtering features [Nadeu *et al.* 1996] and on hybrid HMM/ANN models, has been trained for both Spanish and French (these last models are the ones used in [Zamora-Martinez *et al.* 2012]), although the experiments reported in this work are limited to HTR.

17 This value is not very critical, he use a value between 500 or 1000 which produce an barely noticeable overhead and allows the system to work before all hypotheses have been generated. The histogram size value usually varies between 2000 and 25000.

18 Otherwise the use of threads would be useless!

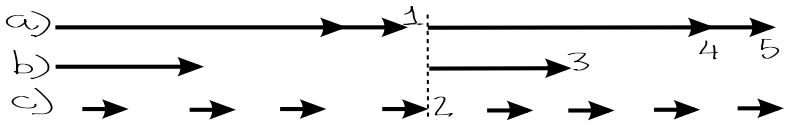


Figure 169: Overlapping and synchronization of threads in the one pass decoder without dictionary look-ups: a) lexicon decoder, b) LM and trellis updating, and c) computation of following pruning threshold. 1) the lexicon decoder finishes a decoding stage, 2) the pruning threshold is obtained, in this moment the lexicon decoder can send the set of hypothesized words to the LM and trellis updating thread, 2) LM and trellis updating is complete, between 2) and 4) the lexicon decoder processes all lexicon hypotheses excepting those who are children of the root, in 4) the lexicon decoder waits until 3) is finished, but usually LM updating is much faster than lexicon decoding, between 4) an 5) the lexicon decoder processes hypotheses which are children of the root. We can observe in c) that the pruning thread processes the new lexicon hypotheses very fast and usually have to wait, but this thread contribute to minimize the latency between 1) and 2).

by the LM and trellis look-up, since this result would not be needed until the nodes which are children of the root of the tree lexicon are to be processed (at the end).

One of the main advantages of this system is that parallelization comes without penalization, that is, tasks are divided in a very natural way, no redundant computation is done and synchronization overhead is nearly nonexistent. The only drawback of this design, as is, it that it does not scale well with an increasing number of cores, since the number of threads is fixed. We have measured the CPU usage of this decoder obtaining up to 200 percent with tree lexicons without LMLA and up to 160 percent with LMLA on ASR tasks.

Another important advantage is memory usage since only two pair of vectors which can be given a bounded size suffices to deal with all lexicon hypotheses, as described in Section 10.6.9. The previous decoder based on tree copies required a separate data structure for every "active LM state".

The most obvious way to scale the current approach to as many cores as available consists of statically splitting the lexicon tree into several sub-trees (and to associated a different lexicon decoder, and hence a different thread, to each of them.

12.3.2 Dealing with silence/space models

This section briefly describes how to deal with initial and final silences as well as with inter-word silences, which is an ad hoc but very important for both ASR and HTR tasks. although HTR would employ spaces, the use of the term "silence" is clear.

The approach used by the DAG generation algorithm in the two stage decoder to deal with optional initial/final/inter-word silences, was to use a transducer to control its behavior, as described in Section 11.1.5 were the use of optional spaces and the context dependency modeling were proposed as examples. We could have implemented a

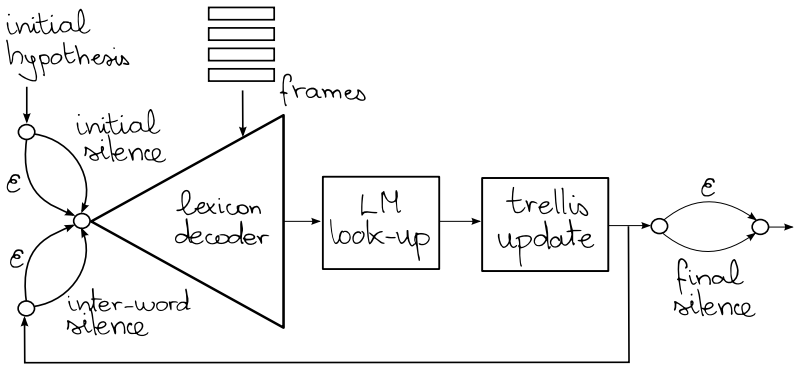


Figure 170: One pass decoder modified to deal with optional silences.

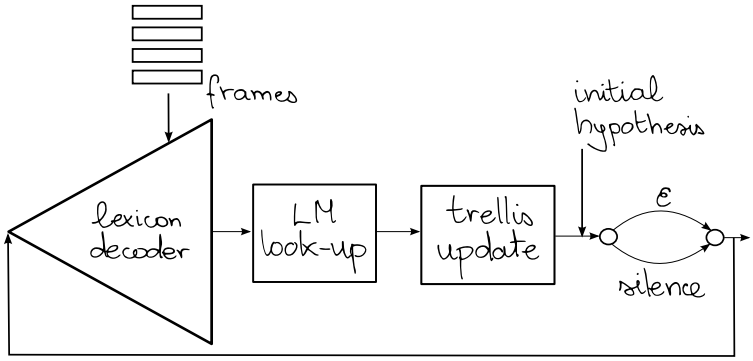


Figure 171: Scheme of a one pass decoder similar to that of Figure 170 but modified to deal with optional silences specialized for the case when initial, inter-word and final silences use the same model.

generic solution for the one pass decoder instead of the ad hoc solution proposed here, but it seems overkill and very tedious to implement.

Figure 170 shows the scheme that can be used to deal with optional silences.[19] Silences are not taken into account by the LM, although it would be easy to modify the proposed scheme to act differently. The initial hypothesis associated to the initial LM state and the initial trellis node is used by the initial silence model. Hypotheses outgoing the lexicon decoder are updated with LM information and are then tracked by the trellis no matter if they are used to feed the lexicon decoder (in that case, there is an optional inter-word silence) or used, at the end, to obtain the output. In the last case, they are updated by the LM using the probability of the LM state of being final.[20] Let us remark that, since we do not know the duration of the final silence, hypotheses outgoing the final silence are only consulted after the last frame has been consumed.

19 They are optional thanks to the null transitions depicted in the scheme.

20 Which is the same as transiting from this LM state using the final context cue. This could be done before or after entering an optional final silence. Each option has its pros and cons. It is probably better to apply this LM update at the end.

Silence and space models usually have topologies quite different from the typical phoneme or grapheme models (see Section 7.3.1). Thus, one of the advantages of this scheme is that the silence models are not taken into account by the lexicon decoder and this makes easier to use specialized lexicon decoders.

It is possible to simplify this scheme when the same HMM model is used to model the three types of silence, as shown in Figure 171. We have used this last option when implementing our ASR dictation tool.

12.4 DEALING WITH EXTERNAL OVER-SEGMENTATION

The use of over-segmentation by the two stage decoder is addressed in Section 11.1.4. This section describes how this information can be used in the one pass decoder scheme we are describing.

Over-segmentation information may be very diverse but, in this work, we restrict to some type of general tags which are associated to frame boundaries (see Section 3.1.4):

- the frame boundary may have a word transition;
- the frame boundary cannot have a word transition;
- we can assure the frame boundary correspond to a word transition;
- this frame boundary begins a "wide sure boundary", meaning that a word transition has to be produced inside this wide boundary or set of contiguous frame boundaries,
- this frame ends a "wide sure boundary".

The absence of over-segmentation information is similar to apply always a default value to allow a word change in every frame boundary. In order to forbid a word transition in a given boundary, assuming this transition is a word ending, we can ignore (or to specialize the Viterbi step in order not to produce them) the hypotheses from the lexicon decoder to the LM updating stage. In order to forbid the creation of words starting in a given boundary, we ignore the hypotheses entering the lexicon decoder after this boundary (associated to the following frame). Since our over-segmenter does not distinguish both cases, we perform both operations.

The possibility of "sure boundaries" is a feature less trivial but also easy to implement. Indeed, this can be seen as a particular case of a "wide sure boundary" of length 1. It is thus better to describe the general case, although this one is more easy to catch and may help to better understand the general case: if we remove all active hypotheses inside the lexicon decoder and inside silence models, only the hypotheses which has just been generated at the output of the lexicon decoder remain alive so that the set of final solutions only contain hypotheses which have produced a word transition in the current frame boundary.

In order to implement the "wide boundary sure word frontier" we have to forbid any hypothesis generated before the wide boundary

region to be alive after this region is finished. We employ a "mark and sweep" technique similar to those used by garbage collection algorithms. At the beginning of the wide region (in the first boundary) we simply mark all hypotheses. We have reserved one bit of the hypothesis representation for that purpose. The lexicon decoder simply ignores this bit which is copied from one active state to the other. This mark is removed from all hypotheses at the output of the lexicon decoder. In this way, when a new hypotheses is fed to the lexicon decoder, the bit is unmarked. At the end of the wide boundary region we remove all marked hypotheses so that only those who have been generated in this region survive.

12.5 SUMMARY AND SOME CONCLUSIONS

The purpose of this chapter is twofold: On the one side, to provide a dataflow module that can make use of a LM and that receives an input DAG as those generated by the `seq_graph_gen` of the previous chapter. On the other side, to describe a one-pass decoder that can use the same LM of the previous DAG decoder and the same lexicons as the `seq_graph_gen`.

The particularity of this one-pass decoder is that it combines the capability of the lexicon decoder to manage multiple hypotheses as well as the bunch mode of the LM interface described in Section 9.1 in order to achieve an overall system where these parts can work quite independently most of the time. It is possible to parallelize this one-pass decoder into multiple threads that are dedicated each one to a different component including, besides the previously mentioned ones, a module for dealing with the preprocessing and another one for computing the (scaled) emission likelihoods that are required by the lexicon decoder. The main limitation of this parallelization is that the number of threads cannot scale to an arbitrary value. It is possible to further parallelize this decoder, nevertheless, by splitting the lexicon decoder and the LM estimator into parts.

A notable extension of these decoders is the capability of using information provided by an external oversegmenter. The use of this information, for the two-stage decoder, is placed in the DAG generator as was described in Section 11.1.4 from previous chapter. Relating the one-pass decoder, we are not aware of any previous description of the inclusion of such a system in the literature.

As future work, we plan to extend the above decoders to deal with context free languages by using RTNs and the algorithms proposed in Chapter 5, to adapt the one-pass decoder to use LMLA NN LMs described in Section 9.5.4 and to adapt the models for dealing with interactive transcription using techniques more sophisticated than the ones used so far (described in Section 9.6).

Part III

Experimentation and results

13 LM EXPERIMENTATION

What most experimenters take for granted
before they begin their experiments
is infinitely more interesting
than any results to which their experiments lead.

Norbert Wiener

CONTENTS

The main aim of this chapter is to show some results related to language modeling. It is a complement of Chapters 6 and 9 where the ideas were presented. The structure of chapter is as follows: A first section provides some empirical validation on the assumptions used in Section 9.3.1 to design the automaton representation of count-based n-grams proposed in this work. Three different corpora have been used together with different vocabulary sizes and n-gram orders. The second section shows some results using skipping-NN LMs to improve the technique of precomputing and storing softmax normalization constants, as described in Section 9.5.1. Finally, some preliminary results of the technique proposed in Section 9.5.2 to estimate the softmax normalization constants of NN LMs are provided.

13.1 SOME BRIEF EMPIRICAL SUPPORT TO OUR AUTOMATA REPRESENTATION

As explained in Section 9.3.1, our representation for count based n-grams (e.g. an `arpa` LM) is based on its prior conversion into automaton, as is described in Section 9.2. The key property that has allowed us to improve this straightforward representation is the fact that, as empirically observed, the fan-out of states (number of outgoing transitions) in this type of automata is distributed in a very asymmetrical way. Indeed, most states have a very low fan-out while and only a few ones have a really huge value.

Table 8: N-gram counts and automaton sizes for some n-gram LMs trained with LOB corpus.

counts	2-gram LM	3-gram LM	4-gram LM
1-grams	103,459	103,459	103,459
2-grams	1,038,123	1,038,123	1,038,123
3-grams	—	201,660	201,660
4-grams	—	—	91,450
states	103,459	193,900	262,630
transitions	1,141,582	1,343,242	1,434,692

Let us remember that the interest of this representation is not only to reduce the memory footprint but, rather, to speed-up LM look-ups by using a different type of search depending on the fan-out as described in Section 9.3.1 and implemented in our April toolkit: transitions outgoing states with a sufficiently low fan-out value can be located with a simpler linear search, while interpolation search and, in some cases, a dense representation may be more suitable for larger fan-out values. In all cases, transitions are sorted to allow these search procedures.

The empirical validation of the asymmetry of the fan-out distribution has been tested with three different corpora: The "Lancaster-Olso Bergen" (LOB) corpus, the "Spanish monolingual" part of the News corpus that appeared in the training data of the 2011 edition of the *Workshop of Machine Translation* (WMT2011) and, finally, the more recent "One billion word benchmark". Two different smoothing techniques have been tested (modified Kneser-Ney and Witten-Bell) using two different toolkits (SRILM [Stolcke 2002] and Ken-LM [Heafield 2011]) as well as different vocabulary sizes and n-gram orders ranging from bigrams up to 5-grams.

Lancaster-Oslo/Bergen corpus

The Lancaster-Oslo/Bergen (LOB) text corpus [Johansson *et al.* 1986] is described in Appendix A.1.1 and has a practical relevance in our work since a sub-set of their sentences has been used to construct the IAM database: IAMdb is a large writer-independent text line recognition task [Marti and Bunke 2002], described in Appendix A.2.1, that has been used in the handwriting recognition experiments described in Chapter 15.

Different n-gram LMs have been generated using the SRI Language Modeling Toolkit [Stolcke 2002] These models have been used in the experiments of Section 15.2. The LMs have been trained using the Witten-Bell discount (WB) and a 103K dictionary, which corresponds to the full training lexicon. We have used this discount instead of the more common modified Kneser-Ney since WB has led to better PPL in validation (see [Zamora-Martínez *et al.* 2014; Table 5] for more details). Each of the resulting `arpa` LMs has been converted into automaton. The n-gram counts and the automaton sizes, for each n-gram order, can be consulted in Table 8.

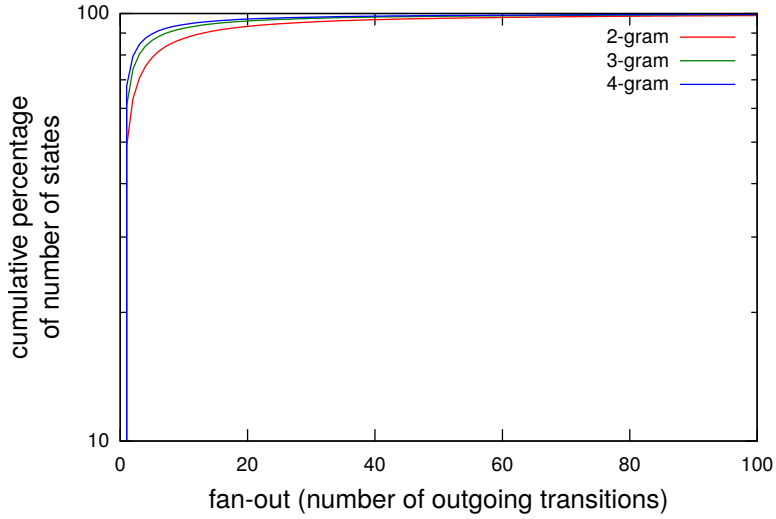


Figure 172: Cumulative percentage of number of states with a fan-out lower than a given threshold for n-grams trained with LOB corpus.

Table 9: Details of some values depicted in Figure 172.

fan-out	cumulative percentage for		
	2-grams	3-grams	4-grams
1	48.92	60.84	67.68
2	63.18	74.14	79.48
3	70.61	80.32	84.67
4	75.44	84.02	87.67
5	78.87	86.50	89.65
10	87.46	92.36	94.23
25	94.61	96.84	97.64
50	97.42	98.51	98.90
100	98.93	99.38	99.54
200	99.59	99.76	99.83

Figure 172 and Table 9 show the percentage of states of each automaton with a fan-out value lower than a given threshold. The plot is truncated at fan-out 100 although the maximum fan-out reached for each model is 103,459 for the three models.[1] We can observe that the vast majority of states have a fan-out lower than 25: 94.6% of bigram states, 96.8% of trigram states and 97.7% of 4-gram states have a fan-out lower than or equal 25, which is a reasonable threshold for changing from linear search to binary search. Indeed, as detailed in Table 9, most of these states have a fan-out significantly lower than 25. We can also observe that this tendency grows faster with the n-gram order. As will be observed below, this tendency also appears when using other corpora.

1 This value is the vocabulary size and the state with this fan-out corresponds to the back-off zero-gram state. The next larger fan-out is considerably smaller (26,325).

Table 10: N-gram counts and automaton sizes for some n-gram LMs trained with the Spanish monolingual part of News corpus from WMT 2011.

counts	2-gram LM	3-gram LM	4-gram LM	5-gram LM
1-grams	151,028	151,028	151,028	151,028
2-grams	9,000,716	9,000,716	9,000,716	9,000,716
3-grams	—	5,985,523	6,566,167	6,624,715
4-grams	—	—	3,119,229	3,358,529
5-grams	—	—	—	819,004
states	151,029	1,610,599	3,230,247	3,887,732
transitions	9,151,744	15,137,267	18,837,140	19,953,992

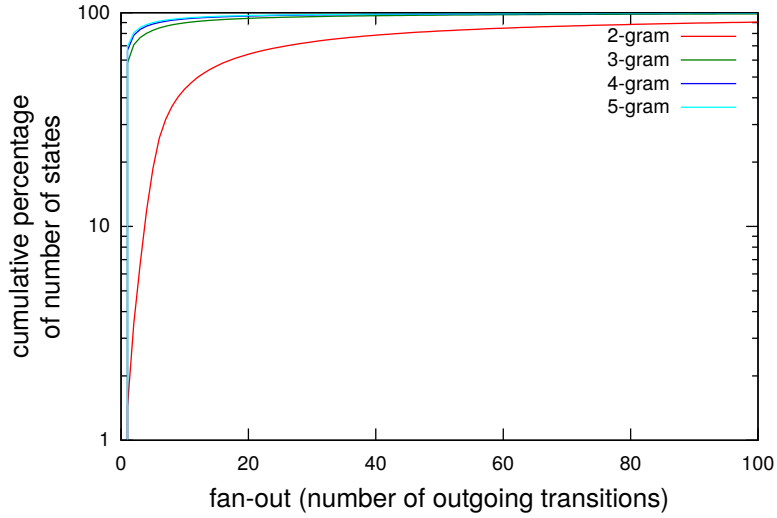


Figure 173: Cumulative percentage of number of states with fan-out $\leqslant$ threshold for the Spanish monolingual part of News corpus from WMT 2011. Only states with a fan-out lower than 100 are plotted.

Spanish monolingual News from WMT2011

Another example is the "Spanish monolingual" part of the News corpus that appeared in the training data of the 2011 edition of the *Workshop of Machine Translation* (WMT2011) [Callison-Burch *et al.* 2011]. Some n-grams (from bigrams to 5-grams) have been trained for this corpus using SRILM toolkit [Stolcke 2002] and the modified Kneser-Ney smoothing. The vocabulary size is around 151K words and both the n-gram counts as well as the resulting automaton sizes are shown in Table 10. As with the previous case, we have also plotted the cumulative percentage of states with a fan-out lower than a given threshold (Figure 173 and Table 11). We can observe the same tendency and properties as with the previous example. Nevertheless, in this case, the percentage of states with a lower fan-out grows in a slower way in the bigram LM.

Table 11: Details of some values depicted in Figure 173.

fan-out	cumulative percentage for			
	2-grams	3-grams	4-grams	5-grams
1	1.43	58.30	66.76	69.42
2	3.51	70.78	78.51	80.67
3	6.71	76.52	83.55	85.38
4	11.83	80.10	86.51	88.09
5	18.69	82.79	88.57	89.96
10	44.00	89.67	93.54	94.39
15	56.36	92.52	95.48	96.10
20	63.94	94.11	96.52	97.02
25	69.18	95.14	97.18	97.59
50	82.13	97.46	98.59	98.81
100	90.39	98.76	99.34	99.44
200	95.10	99.42	99.69	99.74
1000	99.29	99.92	99.96	99.97

n-gram	count
1	335,049
2	36,089,845
3	53,120,562
4	33,464,985
5	24,287,056

Table 12: N-gram counts for a 5-gram (One billion word benchmark). Automaton size:

states	41,777,587
transitions	147,297,497

Table 13: Details of some values depicted in Figure 174.

fan-out	1	2	3	4	5	10	25
%	73.6	84.9	89.5	92.0	93.5	96.8	98.8

One billion word benchmark

Let us finish with a huge corpus, the "One billion word benchmark" [Chelba *et al.* 2013], described into more detail in Appendix A.1.2. This corpus has been used in some experiments described in Section 13.3.

A 5-gram model has been trained for this corpus using the ken-LM toolkit [Heafield 2011]. The lexicon size is around 335K words. The n-gram counts for the trained LM in `arpa` file format, along with the size of the resulting automaton in `lira` format, are shown in Table 12. As with previous examples, we have plotted the cumulative percentage of states with a fan-out lower than a given threshold (Figure 174 and some details in Table 13). As can be observed from this figure, the distribution of fan-outs in this model is similar to the previous LMs. In this case, 73.6% of states have fan-out one, more than 90% of states have a fan-out lower than 4 and only 1.2% of states require a binary search when assuming that a fan-out 25 is used as a threshold to change from linear search to binary search.[2]

2 Note that this value is not hard-wired in the `lira` format and can be easily specified when loading (even with memory mapping) the LM.

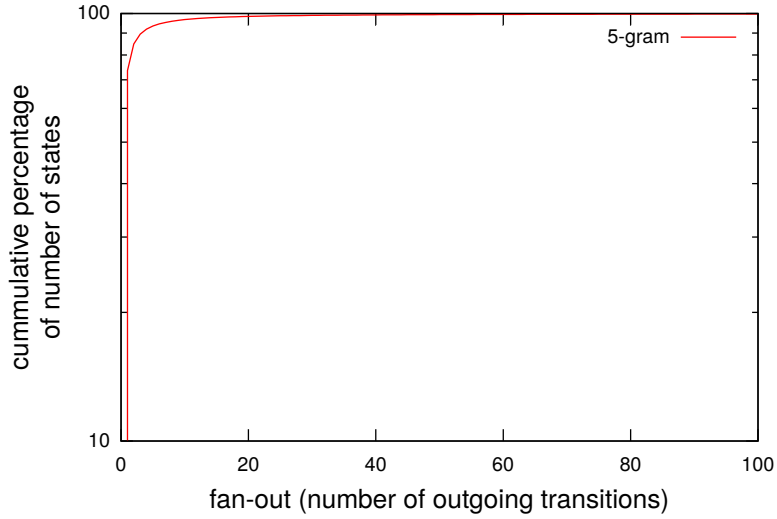


Figure 174: Cumulative percentage of number of states with fan-out lower than a given threshold for different n-gram LMs trained with the One billion word benchmark. Only states with a fan-out lower than 100 are plotted.

We can conclude this section by observing that the assumption required by the efficient use of the proposed `lira` format holds for different corpora and different settings (lexicon sizes, n-gram order, smoothing and LM toolkit). Although an exhaustive trial of the different combinations have not been performed, several examples with a similar and consistent result should be enough to validate the proposed technique. This study could have also been improved by studying the fan-out of states used in practice during decoding.

13.2 FALL BACK SKIPPING NN LMS

This section describes some empirical evidence to support Fall Back Skipping NN LMs (FBS-NN LMs) described in Section 9.5.1. The purpose of these models is to simplify the speed-up technique based on precomputing softmax normalization constants (FB-NN LMs). The idea is to estimate several n-gram orders with a unique NN by using the ⟨NONE⟩ symbol in zero or more input positions (Figure 117). This experimental work has been made in collaboration with Adrián Palacios and Francisco Zamora [Palacios-Corella *et al.* 2014].

13.2.1 Emulating lower order n-gram NNLMs

Let us test the capability of NN LMs with skips to emulate lower order n-grams. Firstly, we will investigate these emulation capabilities, and, secondly, the equivalence between FB-NN LMs and FBS-NN LMs.

Table 14: Lines and words of the News-Commentary corpus (English part).

Set	# lines	# words
Training (News-Commentary 2010)	125.8K	2.9M
Validation (News 2008)	2.0K	49.7K
Test (News 2010)	2.5K	61.9K
Total	130.3K	3.0M

Table 15: Size of the tables of precomputed softmax normalization constants for each n-gram order used in the experimentation.

n-gram order	2	3	4	5
table size	20K	650K	1.79M	2.42M

Corpus

The proposed experimentation is based on the English part of the News-Commentary 2010 corpus, and test sets of 2008 and 2010 editions of the *Workshop of Machine Translation* [Callison-Burch *et al.* 2008; 2010]. Table 14 shows some statistics of this database used to train the LMs. The vocabulary of the training set consists of $|\Omega| = 38,793$ words. The shortlist vocabulary used at both the input and output of the NN LM comprises the $|\Omega'| = 20,000$ most frequent words.

Training the Neural Networks

The NN LMs used as baseline for our experiments are fully described in [Zamora-Martínez *et al.* 2009a; Zamora-Martínez *et al.* 2014]. These models, that have also been used in some HTR experiments described in Section 15.2, will be extended to FBS-NN LMs as described in Section 9.5.1. Backpropagation algorithm and L2 regularized cross-entropy loss function are used to train the NN LM and the FB-NN LM. Three different NNs, for each n-gram order, were trained in order to be linearly combined to compose the final NN LMs or the final FB-NN LMs. These networks have a projection layer of 128, 160, and 208 neurons, respectively. All of them have a hidden layer with 200 neurons. These numbers are based in previous experimentation, and were selected to improve PPL on validation set.

Let us remember that, in order to train a FB-NN LM, the input is stochastically perturbed by the introduction of $\langle\text{NONE}\rangle$ symbol in zero or more input positions. The number of $\langle\text{NONE}\rangle$ symbols is stochastically sampled from a uniform distribution.

After training the networks, the tables of precomputed softmax constants are calculated for both approaches. In both cases, $N - 1$ tables are required: While a different NN is used for each n-gram order in FB-NN LMs, the same NN is used for all orders in the FBS-NN LM approach (only changes the use of the $\langle\text{NONE}\rangle$ to fill the skipped words which are the farthest ones in the LM context). The sizes of these tables, for each n-gram order, are shown in Table 15.

Table 16: PPL measures for the News-Commentary validation and test sets. For the FBS-NN LM, the PPL has been computed for 0 skips (that is, for a 5-gram LM), and for skip combinations which simulate a lower order model. Notice that none of these results are computed using the fall back method, so they are "exact" NN LM results.

	Validation set PPL				Test set PPL			
	n-gram order				n-gram order			
Model	2	3	4	5	2	3	4	5
Bigram NN LM	416	–	–	–	412	–	–	–
Trigram NN LM	–	347	–	–	–	339	–	–
4-gram NN LM	–	–	328	–	–	–	322	–
5-gram NN LM	–	–	–	319	–	–	–	313
FBS-NN LM	424	358	340	334	417	350	333	327

Experiment: Emulating Lower Order n-grams with Skipping NN LM

In order to evaluate the ability of FBS-NN LM to simplify the FB-NN LMs first presented at [Zamora-Martínez *et al.* 2009a], Table 16 shows PPL of the validation and test sets obtained by using NN LM of different n-gram orders and one 5-gram FBS-NN LM, where the skipping positions are set to model equal or lower orders. Values of the first rows are the PPL measures for each one of the NN LM. The last row contains the PPL values for the FBS-NN LM emulating $N-1$ LMs. The last value of this row, for each set, is the PPL for the set without perturbing its input, that is, a 5-gram language model. The previous values are obtained after using skipping configurations that emulate a lower order n-gram LM. For example, the PPL of the validation or test set for the 4-gram LM with the FBS-NN LM is obtained after using the skipping mask that replaces the furthest word from the context by ⟨NONE⟩. Notice that none of these results are computed using the fall back method, so they are "exact" NN LM results. Out of shortlist (OOS) probabilities are smoothed using a unigram over all OOS words.

We can observe that the PPL values for each one of the n-gram NN LM are slightly better than the corresponding of the FBS-NN LM. Nevertheless, we will observe in Section 13.2.2 that these differences are not significant enough to harm the performance of a SMT system.

Experiment: Comparing Fall Back NN LM and Fall Back Skipping NN LM

Let us now compare the PPL for the validation and test sets given by both approaches (FB-NN LMs and FBS-NN LMs). For this experimentation, a count-based 4-gram has been trained with SRI toolkit [Stolcke 2002]. SRI has been configured to use modified Kneser-Ney smoothing with interpolation and unknown word probability computation. It has been linearly combined with either the FB-NN LM or the FBS-NN LM. The linear combination weights are optimized to minimize the PPL in the validation set. The PPL measures, shown at Table 17, exhibit no more that 5 points of difference between the two schemes.

Table 17: PPL measures for the News-Commentary validation and test sets, given by SRI count-based n-gram model, and the proposed FB-NN LM and FBS-NN LM models linearly combined with the 4-gram count-based model.

	Validation set PPL			Test set PPL		
n-gram order	SRI	FB-NN LM	FBS-NN LM	SRI	FB-NN LM	FBS-NN LM
2	332	252	253	409	245	245
3	308	235	239	377	228	231
4	305	231	235	297	224	228
5	305	230	235	297	223	227

Table 18: Statistics of the bilingual Spanish-English part of corpus News.

	Spanish		English	
Corpora	# lines	# words	# lines	# words
News2008	2.0K	52.6K	2.0K	49.7K
News2009	2.5K	68.0K	2.5K	65.6K
News2010	2.5K	65.5K	2.5K	61.9K

13.2.2 Models in a machine translation system

Corpus

The experiments were performed with the corpus of the News-Commentary 2010 Spanish-English task, from the *2010 Workshop of Machine Translation* (WMT'10) [Callison-Burch *et al.* 2010]. Statistics[3] extracted from this corpus are shown in Tables 14 and 18. The English vocabulary was extracted from the 80.9K sentences with lengths up to 40 words. The News2008 set was used as a development set; the News2009 set was used as an internal test set, for comparison purposes between systems. Finally, the News2010 set was used as a final test to measure the generalization ability of the full experimentation.

Training the Neural Networks

All translation systems were trained from Giza++ [Och and Ney 2003] word alignments using the heuristic `grow-diag-final-and`. Relating n-gram LMs, they were trained with SRI tookit [Stolcke 2002], and the rest of models using our `April` toolkit. Phrase-based system combines 14 models, the standard configuration of the open-source machine translation toolkit Moses [Koehn *et al.* 2007], in its standard setup. All systems were optimized using the MERT procedure on the News2008 set. The phrase-based system was trained with Moses, and after tuned and run using Moses and `April`. Table 19 shows the obtained baseline performance for the News2010 test set, along with the average time to decode each sentence.[4]

3 All numbers are computed after cleaning, tokenization and lowercase preprocessing. The tokenization was done using the script `tokenizer.perl` from the WMT'10.

4 An Intel(R) Core(TM) i5 CPU 750 @2.67GHz CPU with 16GB RAM has been used to measure decoding times.

Table 19: Baseline results using Moses and `April` decoders for the News2010 test set using a count-based 4-gram as target language model. Time is the average decoding time per sentence. All the numbers are computed over lower-cased and tokenized sentences.

System	BLEU	TER	Time (s/sentence)
Moses	22.6	57.8	0.6
April	22.7	57.8	0.4

Table 20: BLEU and TER for the News2010 test set for the two tpyes models: FB-NN LM and FBS-NN LM, for different n-gram orders. Time is computed as the count-based translation time plus the rescoring step time. All the numbers are computed over lower-cased and tokenized sentences.

	BLEU %		TER %		Time
n-gram order	FB-NN LM	FBS-NN LM	FB-NN LM	FBS-NN LM	(s/sentence)
2	22.9	22.9	57.5	57.5	0.5
3	23.2	23.1	57.4	57.4	0.5
4	23.3	23.2	57.3	57.4	0.6
5	23.2	23.1	57.4	57.4	0.6

Translation Experiments

The NN LM are used to re-rank the N-best hypotheses generated by the search of `April` (in this experiment, $N = 1000$). The LM generated by each type of model is compared for each n-gram order. Table 20 shows the BLEU and TER of the translations given by each system, after re-ranking, along with the decoding times, computed as the average per sentence of the count-based translation plus the rescoring step time with the FBS-NN LM. Note that the obtained values are almost equal for both types of models. If we compare with the baseline results, significant improvements are observed under a "pairwise comparison" test [Koehn 2004] with 95% confidence. Regarding the decoding times, the proposed LMs are very efficient for rescoring. Table 21 shows the performance and decoding times by using the exact[5] NN LM models. Table 21 shows negligible differences between both fall back models and exact NN LM, but a very important difference in decoding time which becomes larger when increasing the n-gram order.

Since there are no significant differences have been observed between FB-NN LM and FBS-NN LM, for the proposed task, we can conclude that the use of FBS-NN LMs does not alter the performance while being much simpler and faster to train.

The proposed models allows us to integrate NN LM into the decoding stage, instead of rescoring N-best lists, as was shown at [Zamora-Martínez *et al.* 2010]. Further experiments should be done to estimate the advantage of integrated decoding vs. N-best lists rescoring. Another line of future work would be the integration of "bunch" queries during evaluation, which is easier when using FBS-NN LMs.

5 When a softmax constant is not found at the precomputed table, it is computed exactly and memoized for future use during decoding of the same sentence.

Table 21: BLEU and TER for the News2010 test set for "exact" NN LM models. Time is computed as the count-based translation time plus the rescoring step time. Notice that this models use a table of constants to increase its evaluation speed, but when a constant is not found it is computed and memoized while decoding the same sentence. All the numbers are computed over lower-cased and tokenized sentences.

n-gram order	BLEU %	TER %	Time (s/sentence)
2	22.9	57.5	0.5
3	23.3	57.3	0.7
4	23.4	57.2	1.5
5	23.2	57.3	2.6

13.3 NN LM SOFMAX ESTIMATION WITH AUXILIARY LMS

This section describes some empirical evidence to give support to the ideas presented in Section 9.5.2 to estimate the normalization constant of a softmax output by assuming certain correlation or similitude between the output of the NN LM and the corresponding result of an auxiliary LM, on the same words, for a reduced subset of vocabulary. In this way, the unnormalized outputs produced by the NN LM are used, together with the outputs of the (less expensive to compute) auxiliary model. Two different techniques were described in Section 9.5.2 based, respectively, in minimizing a loss function and in aggregating several outputs into one or several classes or categories. This experiment has been tested on the same combination of LOB, Wellington and Brown used in the experiments of Section 15.2. After training a 4-gram NN LM and an statistical or count based auxiliary model, we have estimated the softmax normalization constant and its true value for a set of n-grams appeared in a test subset. Figure 175 shows the relative error of the softmax normalization for these look-ups in the form of an histogram of frequencies. In other words, we can observe how frequently is each possible relative error. We can observe that both approaches obtain quite reasonable and promising results and that, at least for this particular task, the aggregation approach outperforms the former alternative based on minimizing the MSE loss.

It would be unfair to measure the PPL when using the softmax estimation. Another obvious technique to evaluate the goodness of the estimation is by means of extrinsic measures such as the WER when the LM is used in a ASR or HTR task or, alternatively, as in previous section, the BLEU and TER when these models are used in a SMT task. Relating possible comparisons with alternative approaches, this approach could be compared with the naive assumption that the softmax is a constant. This assumption may make more sense when the training is performed with an explicit regularization to reduce the variance of the softmax. It would be also interesting to compare the proposed technique with the memoization of normalization constants discussed in previous section.

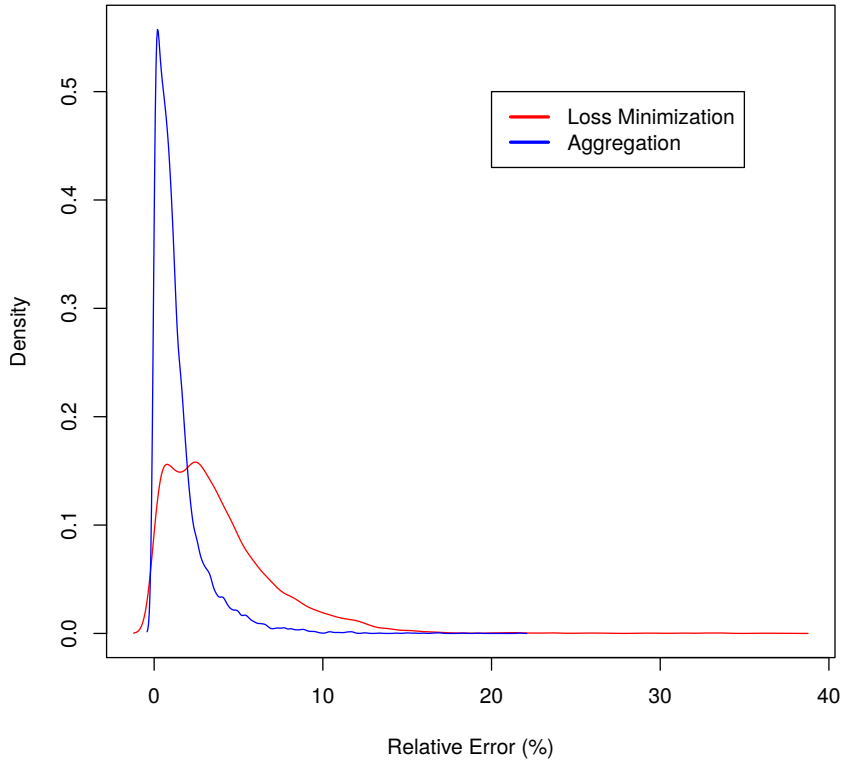


Figure 175: Comparison of the distribution of the relative error w.r.t the true softmax computed constant when using the loss minimization and the aggregation approaches. These results have been obtained with the NNLMs trained with the LOB corpus and described in Section 13.2. As can be observed, although both approaches display good preliminary results, the aggregation approach seems to outperform loss minimization.

13.4 SUMMARY AND SOME CONCLUSIONS

This chapter complements Chapters 6 and 9, specially the second one since some of the techniques proposed there have been empirically validated here.

The first contribution is just to show some statistics about the actual fan-outs used by several LMs trained on different non-trivial corpora. The LM representation proposed in Section 9.3.1 assumed that most states have a very low fan-out while only a minority have a large fan-out value. The purpose of the first section of this chapter has been to provide some empirical support of this assumption. We have seen how this property holds for different corpora and settings (lexicon sizes, n-gram order, smoothing and LM toolkit). Although an exhaustive trial of the different combinations have not been performed, several examples with a similar and consistent result should be enough to validate the proposed technique.

Our study could have been improved by studying the fan-out of states used in practice during decoding. Instead, we made some preliminary experiments, prior to the construction of this format, to deter-

mine the thresholds to choose the proper search technique. Although this is usually hardware dependent, we can roughly say that linear search is usually faster until reaching fan-outs over 25 and, for this value, more than 90% of states fall into this first category. It is not so clear when interpolation search outperforms binary search. Since interpolation search depends more on the distribution of data than on the vector sizes, we should have measured this effect in our experiments. Another missing experimentation is the study of the improvements obtained by using the bunch search technique proposed in Section 9.1.2. One of the main reasons to support this idea is the assumption that cache misses constitute a noticeable bottleneck in LM searches.

The second section of this chapter is an empirical validation of the Fall Back Skipping NN LMs. This part is a collaborative work with Adrián Palacios and Francisco Zamora. This experimentation shows that they are very well suited for emulating lower order n-grams. Nevertheless, more experimentation is required to further assess the usefulness of the more general setting described here.

Finally, the last contribution is a preliminary study of a technique proposed in this work to avoid the computation of the softmax normalization constants by estimating it with the aid of an auxiliary LM. Some preliminary experimental measurements are presented.

The most clear future work assignment is to further continue and develop this experimental part related with the novel NN LMs ideas.

14 HTR PREPROCESSING

THE BEAUTY AND NOBILITY,
THE AUGUST MISSION AND DESTINY,
OF HUMAN HANDWRITING.

George Bernard Shaw, Pygmalion

CONTENTS

Unconstrained handwritten text recognition (HTR) remains a difficult task due to several reasons including, among others, the high variability of writing styles and the problem of out of vocabulary words. Roughly speaking, HTR is usually classified into two main areas depending on the way input data is acquired, as explained in Section 2.7. Online HTR is devoted to handwriting acquired with special devices which capture the trajectory of the stylus while the user is writing, whereas offline HTR handles images of previously acquired text. This chapter is focused on offline preprocessing and feature extraction, although the online preprocessing is also briefly covered, mainly to explain how the proposed offline preprocessing has been adapted to online modality. Next chapter presents some experimental results on HTR which empirically validates the techniques described here.

Handwritten image normalization from a scanned image comprises several steps, which usually begin with image cleaning and page skew correction followed by layout analysis in order to detect text blocks,

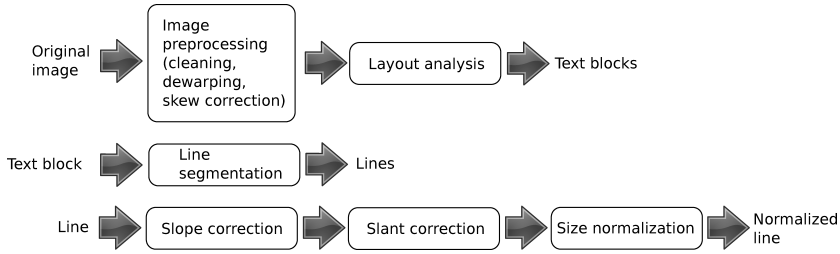


Figure 176: Bird's eye view of handwriting recognition preprocessing stages from the scanned document to preprocessed handwritten lines. Some stages can be skipped (e.g. de-warping) or can vary in position. Stroke thickness normalization is not included in the illustration but it can be applied in different positions of this chain (e.g. after slant removal), replaced by a skeletonization or just skipped (as in our case).

and line detection to separate blocks into lines [Marti and Bunke 2001]. These steps have been briefly covered in Section 2.7 and are illustrated in Figure 176.

Our work is focused on recognizing text lines, which can be treated like sequences even in the offline modality. Also, all experiments have been conducted on a database of skew corrected line images so, for the sake of brevity, skew correction and text line extraction will not be discussed here (they have been explained in Section 2.7) although some techniques proposed in this Chapter could be adapted to deal with line extraction and layout detection.

The main aim of preprocessing consists of reducing the variability while preserving the information useful to correctly classify. Handwritten text lines can be divided in three different areas[1] (ascenders, descenders and the main body area or core region) which can be separated by imaginary reference lines, as shown in Figure 177. In handwriting, the preprocessing applied to a text line usually implies to remove the slope and the slant and also to normalize the size in order to achieve three goals: First, since the normalized text line image is chopped into image columns, we would like that every area is placed in a different set of rows. Second, as the main body area usually contains much more information than ascenders and descenders, it is better to assign more parameters, accordingly, to the main body area, although this, obviously, depends on the type of feature extraction method applied after preprocessing and will be discussed into more detail in Section 14.6. Third, if we cut the image into columns, the same column might contain strokes from different graphemes due to slant. In this way, the purpose of slant correction is not limited to reduce the variability but also to make it more justifiable the suitability of the joint segmentation and classification approach.

Another source of variability is due to the writing instrument, for instance the thickness (Figure 178) or the aspect of strokes. We have removed the variability relating the gray level, but we have not normalized thickness, relying on the statistical learning techniques and the corpus variability.

1 At least for latin derived alphabets.

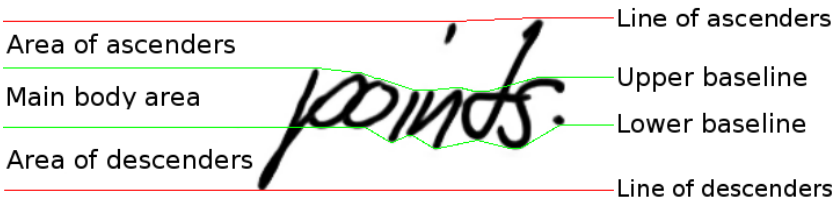


Figure 177: Example of text line image with the different areas (ascenders, descenders, and main body area) and the reference baselines (upper and the lower baselines, and the lines of ascenders and descenders) of the cursive script (from [España-Boquera *et al.* 2011; Fig. 3]).

Figure 178: Handwritten words using devices of different thickness (image preserves relative size).

The particularity of the proposed techniques consists of using supervised learning methods instead of just relying on geometrical heuristics, which corresponds to the ideas proposed in Section 2.9.3.

14.1 IMAGE CLEANING

The first preprocessing step consists of cleaning and enhancing the scanned image in order to remove some noise which may be due to the presence of shadows or defects in the paper, ink bleed-through or even to a non-uniform illumination during the acquisition process. It is even possible to recover some broken strokes, repair ink smears, etc. Image cleaning is particularly important in noisy documents like historical and degraded manuscripts, although even modern and apparently clean documents may benefit from this process.

Since gray level variation among different foreground and background images is not useful to recognize what has been written, many works propose to binarize the image, but we believe it is better to convert them into *normalized* gray level images where intermediate gray values are used to reduce contour aliasing and to avoid hard decisions in case of foreground/background ambiguity. In this way, intermediate gray values are mostly located at the borders of strokes. Besides, these values, when represented between 0 and 1, may be interpreted as the probability of ink.[2]

normalized gray level

Moreover, since image cleaning is one of the very first preprocessing steps and it is usually followed by other stages which transform the

2 Assuming an underlying black and white image composed of strokes which much more precision, the value associated to pixels corresponds to the probability of touching an ink region when taking a random point inside the pixel.

image, it is better to maintain images in gray level during these geometrical transformations. Also, many feature extraction techniques may receive gray level images.

contrast normalization

Note that other works which also use gray level values may follow a completely different approach. For instance, in [Pesch *et al.* 2012] a contrast normalization process is applied where 70 percent of the lightest pixels are mapped to white, the 5 percent of the darkest pixels are converted to black and the rest are linearly scaled. A different technique which does not need to provide estimated percentages of black and white pixels, based on machine learning techniques, is proposed in this section.

There are many different types of techniques in the literature to clean and to binarize the image, among them:

- some convolutional filters are useful to remove additive noise or to smooth the image (mean and Gaussian filters, for instance), other filters, like the median filter, are useful to remove salt-and-pepper noise or also to estimate the background. It is possible to obtain an image mask with median filter and to apply it to the original image in order to clean the background without loosing too much detail at contours;
- morphological dilation/erosion operations or also the computation of connected components in order to detect and to remove blobs or to normalize thickness. Mathematical morphological operations may simplify images while preserving some essential shape characteristics, they can be useful to remove lines from forms [Ye *et al.* 2001];
- binarization has been widely studied in the literature. Most binarization methods are based on thresholding techniques, like Otsu [Otsu 1979] algorithm which computes a global threshold. Others, like Niblack [Niblack 1990], compute a local threshold in an adaptive way using information from the pixel neighborhood;
- this classification is far from being exhaustive, there exists many different techniques based on statistical approaches, as is the case of Markov Random Fields [Wolf 2010] or neural networks as described below.

The solution proposed in this work consists of performing a *unique cleaning and enhancement step by using neural network convolution filters which produces a gray level image*. Neural networks have already been used in previous works for image restoration by learning filters from examples [Stubberud *et al.* 1995; Egmont-Petersen *et al.* 2002; Suzuki *et al.* 2003; Marinai *et al.* 2005]. We have used a neural network filter to estimate the gray level of one pixel at a time as illustrated in Figure 179: a MLP (from now on called Enhancer-MLP) is fed with a square of pixels centered at the pixel to be cleaned, and the output is the restored value of the pixel. The entire image is cleaned by scanning all the pixels in this way. Note that each cleaned pixel is estimated independently. A joint estimation could be more accurate although this simplification is justified by the high correlation of MLP inputs of neighboring pixels which are mostly overlapped.

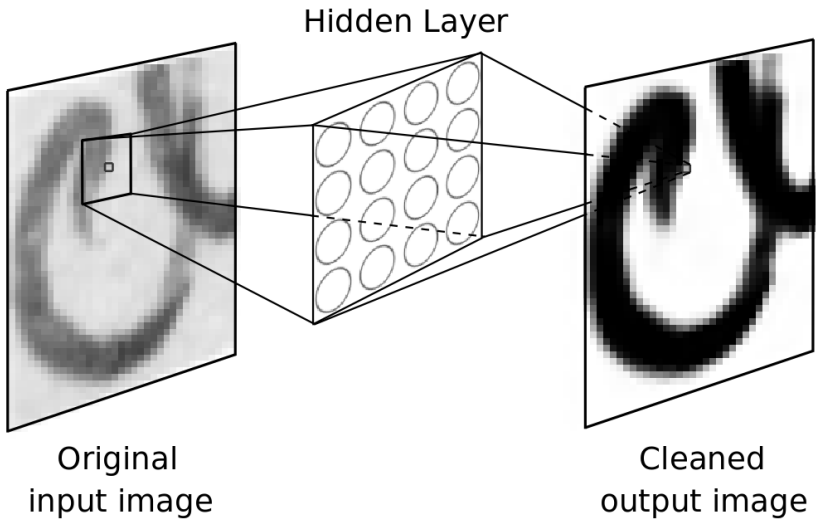


Figure 179: Enhancer-MLP: An MLP to enhance and clean images. The entire input image is cleaned by scanning it with the neural network (figure taken from [España-Boquera *et al.* 2011; Fig. 1]).

supervised training

One of the main advantages of these filters is their capability to learn from examples in order to adapt to the type of local noise to be removed, but this advantage is also a disadvantage since they require supervised training.[3] Some noise images with their corresponding clean counterparts must be obtained to train the filter. We have used line images from the offline IAM database, some of them have been manually cleaned,[4] others have been artificially noised,[5] as we had previously done in [Hidalgo *et al.* 2005]. This supervised data is used to train several MLPs using the online version of the back-propagation algorithm with momentum term and mean square error function.[6] The universal approximation property of a MLP guarantees the capability of the neural network to approximate any continuous mapping [Bishop 1995]. Relating the parameters, increasing the window size surrounding the pixel to be cleaned makes the input size to grow quadratically increasing the computational effort to obtain a negligible improvement once a size of 11×11 pixels is reached, being a good compromise between quality and computational requirements. The best MLP topology which had been used in all subsequent experiments contains two hidden layers of 32 and 16 neurons, respectively, using the logistic activation function. A sole output neuron with linear activation obtains the estimated pixel whose value is cropped to

3 It is also possible to use non-supervised data in a limited way. For instance, by using noised images to perform a non-supervised pre-training as proposed in deep-learning.

4 Not only to remove noise from background but also to correct discontinuities and imperfections in the strokes, as is often the case when using some type of pencils.

5 GIMP image manipulation program (`http://www.gimp.org`) has been used for both manually cleaning the images and for applying several noise filters to clean images.

6 Other error functions also correlated with evaluation measures for this task, such as the F-measure, can also be taken into account.

the $[0, 1]$ interval.[7] Some examples of training images together with a real example of a cleaned image are shown in Figure 180.

In presence of different types of noise or just a non-uniform noise, it is possible to train several specialized MLPs and to combine them. This can be done in a non-supervised way by training a different MLP for every type of noise and by clustering the filters which have the same behavior in some validation set afterwards [Zamora-Martínez *et al.* 2007]. Indeed, the main problem of neural filters is the local nature of the input, so it is reasonable to add additional information to the MLP input with more global information[8] to improve the process. These more costly techniques have not been used in this work because all experiments have been conducted with quite clean datasets where noise was very uniform.

assessment measures

Relating the evaluation of the quality of the cleaned images, three main approaches can be considered [Ntirogiannis *et al.* 2013]: by means of human supervision[9], indirectly by comparing the overall performance of a recognition system and, finally, by comparing the cleaned image with a reference or *ground truth*. In order to measure the impact of this stage in the overall system, it would be required to repeat the entire preprocessing and training stages with different cleaning methods. We have not pursued this experimentation since the proposed cleaning process performed much better that other alternatives at our disposal.

14.2 TRACKING REFERENCE LINES

It is clear that slope and size normalization are closely related to the detection of the different areas of the cursive script (Figure 177). Traditional preprocessing methods [Burr 1982; Bozinovic and Srihari 1989; Vinciarelli and Luettin 2001] obtain a rough estimate of the main body area. This estimate is often obtained by means of horizontal histogram projections which implicitly follows this geometric heuristic:

> *Ink density is higher in pixel files belonging to main body area than in ascenders and descenders.*

Other works propose to estimate the upper and lower contours of the image [Pastor Gadea 2007] after applying the "Run-Length Smoothing Algorithm" (RLSA) [Wong *et al.* 1982] (see Figure 181). Geometrical heuristic methods, however, may fail for short words and, since they are based on statistics about the presence of ascenders and descenders, they may be confused in presence of too much or too few ascenders and/or descenders and are also affected by the presence of long horizontal strokes. Other methods to estimate the main body area

7 Gray scale images are represented as floating point values between 0 and 1.

8 Some experiments have shown the benefits of including the pixel values after applying a median filter with a different neighborhood (which produces a rough estimate of the background image). The inclusion of a rough estimate of the histogram of the noise image surrounding the pixel to be cleaned, as well as other features related to the type of noise or even a texture classification of a wider context seems also very promising.

9 Not only visual inspections or subjective evaluations, but also the detection of features and the use of a defined protocol with double blinding, etc.

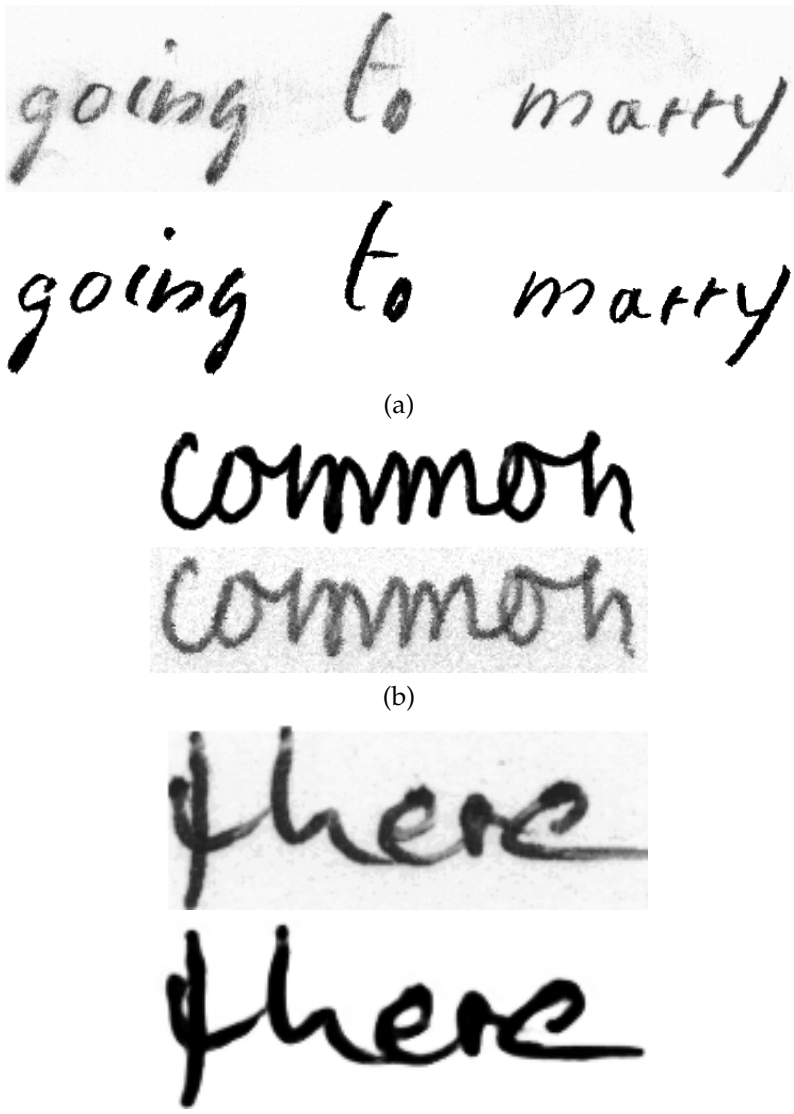


Figure 180: (a) Manually cleaned image, (b) Automatically noised image, (c) Image cleaned with a neural network filter.

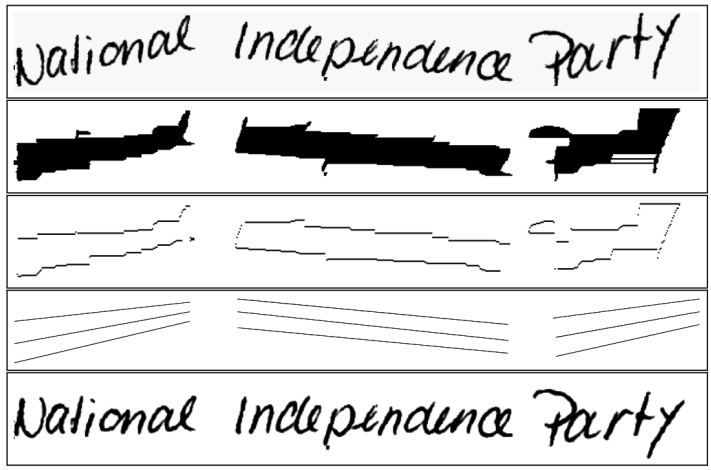


Figure 181: Image borrowed from [Pastor Gadea 2007; Fig. 4.15] showing how to correct the *slope* by approximating the reference lines using the contours of the text image after applying RLSA smoothing.

avoid geometrical heuristics and rely on supervised statistical learning techniques. For instance, [Seiler *et al.* 1996] uses neural networks (Figure 182) to obtain a rough estimate of the main body area. This technique uses this estimate to obtain reference lines by finding local extrema close to the estimated baselines. The use of local extrema can be also found in the literature. For instance, in [Caesar *et al.* 1995] the vertical extreme values are used to estimate the baseline by means of geometric measures, heuristics based on statistics of the presence of ascenders and descenders and regression analysis (see Figure 183):

> *From a topological point of view each vertical extreme value of the contour forms a convex point which may be important for the estimation of the writing lines.*

To our knowledge, none of these methods track baselines and local extrema by classifying those reference points as belonging or not belonging to each type of reference line. We advocate to use automatic learning methods during preprocessing instead of just relying on geometrical heuristics as explained in Section 2.9.3. This approach is not new since it can be traced back, at least, from [Seiler *et al.* 1996]. More recent work based on similar ideas can be found in [Hennig and Sherkat 2002; Simard *et al.* 2005].

Proposed approach

Our approach to image normalization consists of first detecting those local extrema which are classified afterwards by means of supervised machine learning techniques (in our case, neural networks). The use of MLPs is widespread in our preprocessing as can be observed in Table 22 which summarizes the set of networks used in the proposed preprocessing. Figure 188 illustrates the overall preprocessing which is explained below.

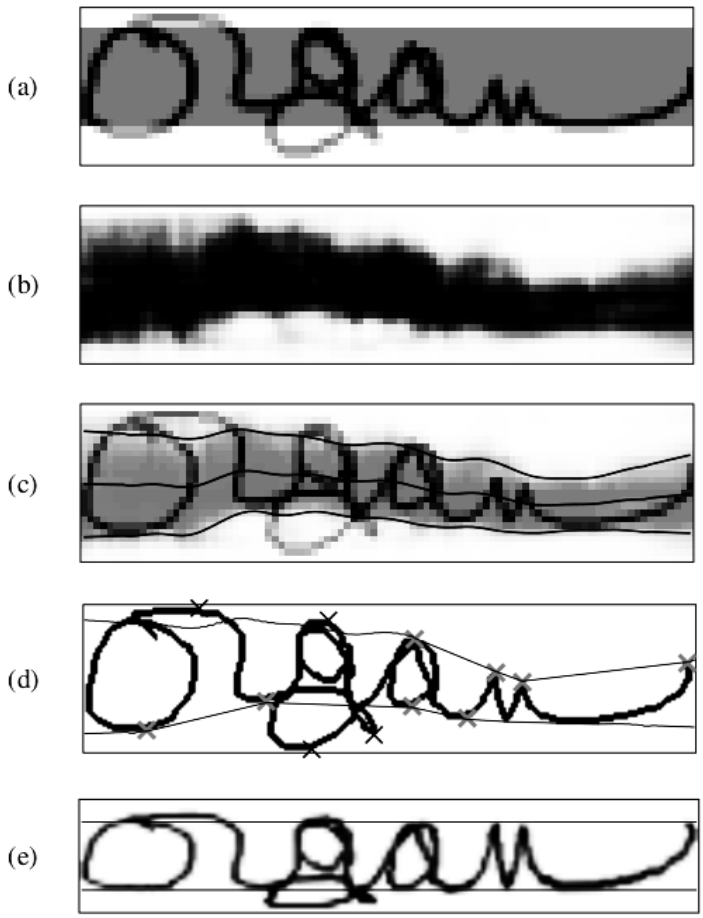


Figure 182: Figure taken from [Seiler *et al.* 1996; Fig. 2] showing a normalization method employing a neural network to obtain a rough estimate of the main body area. From top to bottom: (a) image sampled to a fixed height together with the neural network desired output shown in gray, (b) output of the neural network which traverses the image and estimates where the main body area is, (c) each column is modeled by a Gaussian distribution, mean form the center of the word and three times the standard deviation is a rough estimate of the main body area, (d) local minima and maxima which are close to estimated baselines are used to normalize the image by linear interpolation shown in (e).

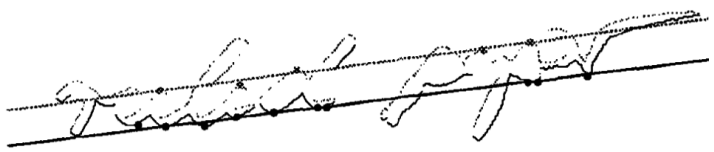


Figure 183: Figure taken from [Caesar *et al.* 1995; Fig. 5] illustrating a baseline estimation technique based con contour processing. The example shows the contour of a skewed line, the set of accepted convex points and the corresponding estimated ruler lines obtained by means of regression analysis.

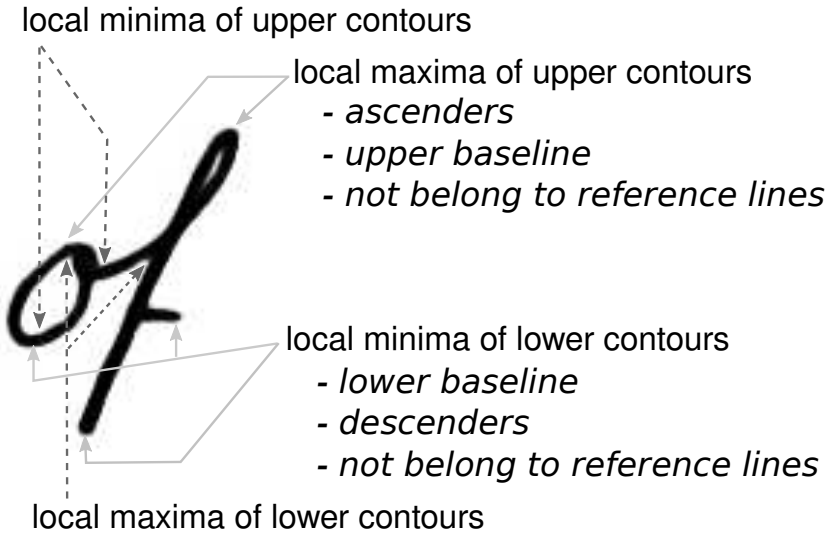


Figure 184: Different types of local extrema. Local maxima of upper contours and local minima of lower contours will be used in this work.

Table 22: MLPs using in the proposed preprocessing. MLPs are used for regression (Enhancer-MLP) and for classification (Slope-MLP, Normalize-MLP, and Slant-MLP).

MLP	Input	Output	MLP topology
Enhancer-MLP	Pixels in a window around current pixel	Restored value of current pixel	11×11-(32-16)-1
Slope-MLP	Pixels in a window around current pixel	Lower baseline/Other class	50×30-(64-16)-2
Slant-MLP	Resized sheared image in a window around current column of pixels	Presence of slant angle	40×40-(64-8)-1
Normalize-MLP	Pixels in a window around current pixel	Ascender/Upper baseline/Lower baseline/Descender/None pixel	50×30-(64-16)-5

14.2.1 Extraction and classification of "interesting points"

Let us see how some types of local extrema, which will be called "interesting points", are obtained and which type of information they can provide relating the reference lines. The use of these points to actually perform slope removal and size normalization is explained in the following sections. As depicted in Figure 184, there are four types of local extrema, two of them will be considered as "interesting points":

- *local maxima of upper contours* can belong to the line of ascenders, to the upper baseline or they cannot belong to any reference line;
- *local minima of lower contours* can be placed in the line of descenders, in the lower baseline or they cannot belong to any reference line.

In order to obtain the local maxima of upper contours we proceed as follows:

1. first, points from the vertical upper contour are extracted. Since a "normalized" gray level image is used, we proceed by scanning the entire image with a window of 3 pixels height and 1 pixel width. The central pixel of this window is an upper contour if the upper pixel is quite bright and the lower pixel is sufficiently dark,[10] although other more sophisticated techniques can also be used;
2. contour points are grouped into lines following a proximity criterion: two pixels on adjacent columns are considered to belong to the same line when the difference between their vertical coordinates is less than 3. Obviously, these thresholds are dependent on the image dpi;
3. finally, the points corresponding to the vertical maxima of these groups are computed. That is, points who have the largest values within a left and a right context, which, for this work, has been set to 6. Note that in the case of a horizontal stroke several points can be considered "interesting".

The local minima of lower contours can be obtained in an analogous way. One feature of this extraction process is the presence of groups of close interesting points, specially where set of local maxima (or minima) are obtained from long horizontal sequences. It is desirable to reduce the number of interesting points to reduce the computational load while preserving the accuracy of the techniques based on them. Although the proposed method to extract these points has demonstrated to be quite robust, [11] we seek to improve the extraction process to make it even less dependent on the image resolution and also more robust facing highly degraded documents.

10 That is, two different thresholds are used by the algorithm. Using the same threshold value would be equivalent to work on the (on the fly) binarized image. Note that the image is not an arbitrary gray level but a "normalized" gray level image where gray values represent uncertainty about the presence of ink, mainly in the borders of strokes.

11 It has been tried with images from different databases with quite satisfactory results.

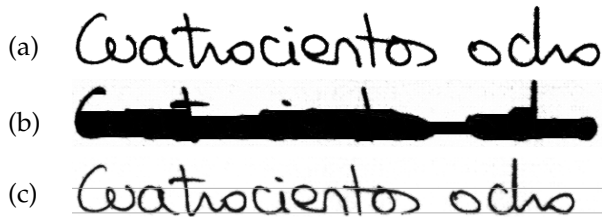


Figure 185: Images extracted from [Gorbe-Moya 2005; Section 3.5] showing the geometric heuristic method used to extract reference lines for the first stage of the active learning approach used in this work: (a) original image (from a Spanish database), (b) image after applying RLSA. A horizontal projection histogram is computed for this image and the first line exceeding 50% of the maximum value from the bottom and from the top are used as estimates of baselines displayed in (c).

14.2.2 Active learning of classes of "interesting points"

Following sections will describe how the classification of interesting points described in the previous section can be used to perform slope correction and size normalization. Let us remember that there are two "natural" classes of interesting points depending on the type of local extrema. In turn, each of these types can be classified into three classes relating reference lines although, if we joint all these types into a sole repertoire only five labels are needed since "none of the previous ones" class appears twice.

A classification of "interesting points" by means of supervised learning techniques, based on neural networks, is proposed in this work. To this end, we need to obtain training and validation patterns. An active learning technique has been used to achieve this goal:

1. first, a subset of 200 line images from the IAMdb training part has been used. Interesting points extracted from these lines have been preliminary classified by comparing their vertical position relating horizontal reference lines[12] (see Figure 185);
2. then, these points have been manually corrected using a graphical tool designed to this end (see Figure 186);
3. this preliminary training set has been used to train a first classifier which has been used to label additional line images, also from the training part of IAMdb;
4. the same graphical tool has been used to validate the previous labeled images.

Finally, 773 lines[13] of the training part of the IAMdb have been obtained using this technique. All interesting point classifiers are based

12 These reference lines correspond, as described in [Gorbe-Moya 2005; Section 3.5] to the first and to the last image rows whose horizontal projection exceeding the 50 percent of the maximum of this horizontal projection. This projection is applied over the image after being applied RLSA smoothing. Points below the estimated lower baseline minus 5% line height are classified as descenders and other points below the estimated lower baseline are classified as belonging to the lower baseline. The distinction between ascenders and upper baseline is performed similarly.

13 The reader is right to find this value arbitrary: 1000 lines were split among several persons until an auto-imposed deadline was reached. More data could have lead to

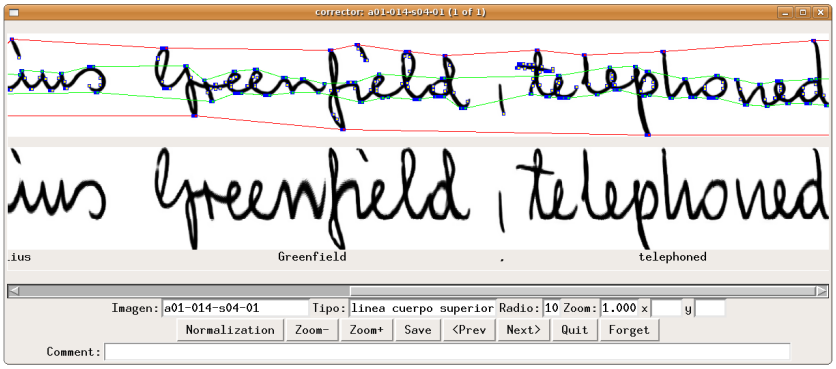


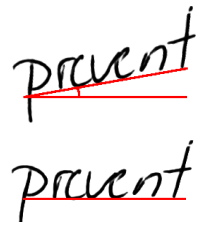
Figure 186: Screenshot of the graphical tool designed to manually supervise the classification of interesting points in order to obtain training data for supervised learning techniques. It is easy to change the label of a point with the mouse and the normalized image is recomputed to better detect possible mistakes.

	Training	Validation
lines	723	50
words	5,249	353
points	430,929	29,965
ascenders	6.08 %	6.09 %
upper baseline	22.13 %	21.87 %
lower baseline	36.01 %	35.74 %
descenders	2.22 %	2.61 %
rest	33.56 %	33.68 %

Table 23: Statistical information about the number of local extrema of each class measured on IAMdb.

on Multilayer Perceptrons (MLPs) which receive at input an image window of 500×250 pixels centered at the point to be classified. This window is down-sampled to 50×30 using a fish-eye distortion (see Figure 187) in order to reduce the input size while preserving a detailed image near the point and a coarse representation of the relative position of the surrounding text. 50 lines have been reserved for validation and the remaining data has been used for training. Table 23 shows some statistics about these sets.

14.3 SLOPE REMOVAL



Slope is defined as the angle between the reference baselines of the text (assuming they are really lines and they are parallel) and the page horizontal direction. Slope is usually corrected by means of a a rotation operation with the same slope angle in the opposite direction, but, in practice, it is accompanied by a vertical translation in order to horizontally align the lower baseline of different image chunks.[14]

As discussed before, most traditional methods to correct slope found in the literature obtain a rough estimate of the main body area [Burr

better classification rate. The overall effort devoted to the entire labeling effort can be measured in some "around 30 hours" distributed among 4 persons during some days.

14 We cannot know if an ink block is a word, part of a word or many words.

Figure 187: Fish-Eye lens example: original image of 500×250 pixels centered at the pixel to be classified (top) and the same image after a fish-eye lens distortion (middle) and the fish-eye distortion downsampled to 50×30 (down).

1982; Bozinovic and Srihari 1989; Vinciarelli and Luettin 2001]. For instance, [Pastor Gadea 2007] describes how to correct the slope by approximating the reference lines using the contours of the text image after applying RLSA smoothing (see Figure 181). We will proceed in a similar way: The image is horizontally divided into segments in order to apply the slope correction to every segment. A vertical histogram projection is used to estimate the mean space width between ink regions, and this value is used as a threshold to split the image into segments. For each of these segments, the lower baseline is estimated and the segment is rotated[15] and vertically translated in order to horizontally align their lower baselines.

There are other slope correction algorithms which do not rely on estimates of the lower contour. For example, some techniques try several slope correction angles in order to choose the angle minimizing an objective function: [Kavallieratou *et al.* 2003] proposes the Wigner-Ville energy as objective function, [Kennard and Barrett 2006] uses the variance, [Pastor Gadea 2007] proposes an objective function based on the maximum of the horizontal projection and the standard deviation.

We propose a technique based on the estimation of the lower baseline but, instead of obtaining a rough estimate of the main body area,

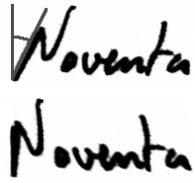
15 When the slope is very severe, a horizontal scaling can be performed to preserve some horizontal alignment with original chunks, risking to increase scale variations.

we propose to classify the interesting points corresponding to local minima of lower contours as explained in Section 14.2. These local extrema may belong to the line of descenders, to the lower baseline or to neither of these lines but, in order to obtain the lower baseline, a binary classifier (lower baseline/other class) suffices. An MLP (this MLP will be called Slope-MLP) with a fish-eye compressed window corresponding to 1500 input neurons has been trained with the database described in the previous section and with the backpropagation with momentum term algorithm. After some experimentation, a MLP topology of three hidden layers of sizes 70, 20, 10 has been chosen. Hidden layers use the logistic activation function, whereas the output layer, with two outputs, uses the softmax activation function.

Once the set of lower baseline interesting points has been determined, a line approximating the lower baseline of every segment is computed by means of least squares fitting of the lower baseline points of each segment.[16]. An example of the splitting process of the estimated lower baseline is shown in Figure 188(d). These lower baselines are used to correct the slope. Figure 188(e) illustrates an example of a slope-corrected image. As future work, the upper baseline can also be estimated in a symmetrical way in order to improve the reliability of the slope angle estimation, but this would be more costly. Other obvious improvements are related to the training of the Slope-MLP by means of an unsupervised pre-training using non-labeled data. It is also possible to include more information in the Slope-MLP input.[17] Another interesting improvement is to perform a joint classification of several interesting points. Is it also possible to produce several preprocessing alternatives when this estimation is not reliable enough in order to delay these hard decisions to later stages, as discussed in Section 2.9.3.

14.4 SLANT REMOVAL

Slant is usually described as the clockwise angle between the vertical direction and the direction of the "vertical text strokes" and may change greatly in handwriting (Figure 189). Slant correction is usually performed by means of a shear operation which transforms the word into an upright position resulting in an image which is independent of this factor. Slant removal reduces the problem of having strokes from several graphemes in the same column, which is a great advantage for recognition systems which decompose the image lines by chopping them vertically.



Many classical deslanting techniques try to estimate the average slant of near vertical strokes, they differ in the way to detect those strokes and stroke angles [Bozinovic and Srihari 1989; Senior and Robinson 1998; El-Yacoubi *et al.* 1999b]. Some works try to directly estimate the stroke angles from their contour. For instance, in [Caesar *et al.*

16 The method is quite robust relating some missclasification of interesting points.

17 Not only a window surrounding the point to be classified but also other data used by other line extraction algorithms such as the horizontal histogram projection, the use of RLSA or even some a priori information of the page layout when available.

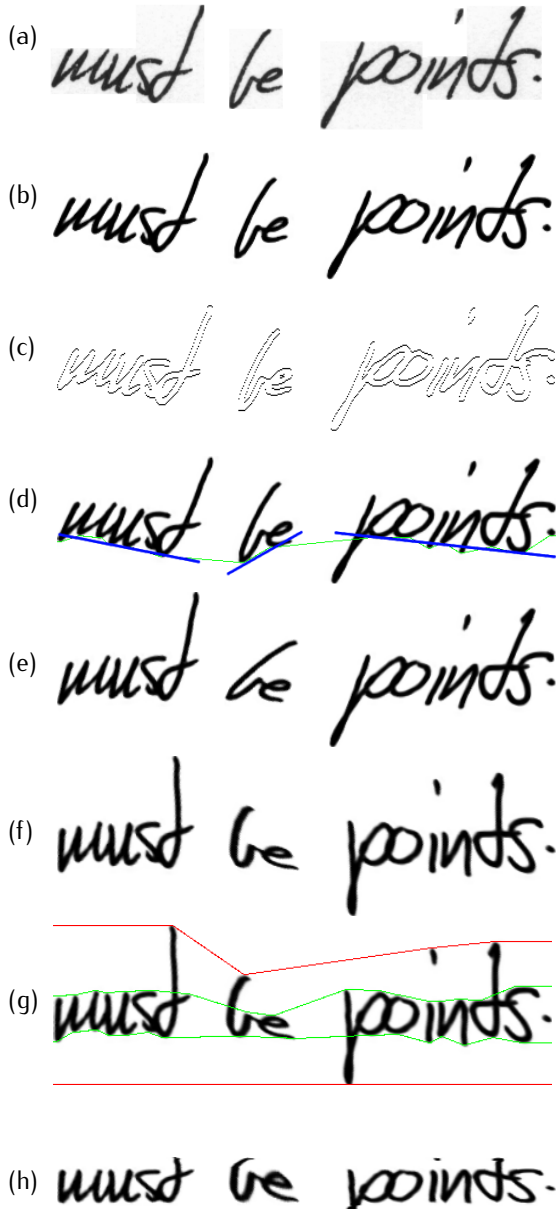


Figure 188: Overview of the entire preprocessing process. From top to bottom: (a) Original image from the IAMdb. (b) Cleaned image. (c) Text contour extracted to obtain local extrema. (d) Local extrema classified by an MLP as lower baseline to be used for slope correction. (e) Slope-corrected image. (f) Slant-corrected image. (g) Local extrema labeled by an MLP as belonging to the four reference lines to be used for size normalization. (h) Normalized final image (figure taken from [España-Boquera *et al.* 2011; Fig. 2]).

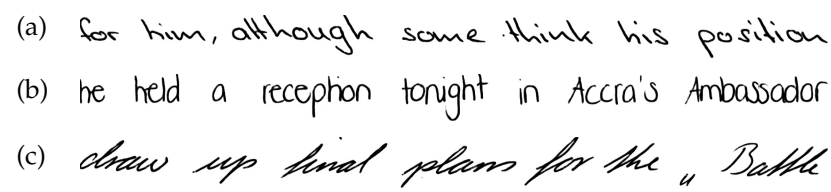
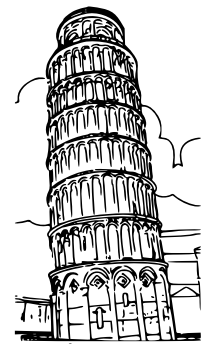


Figure 189: Examples from the IAMdb illustrating different slants angles: (a) text slanted to the left, (b) without slant, (c) to the right.

1993], angles of segments of the contour are weighted by their length. Other methods are based on geometric heuristics relating statistics of deslanted text. For instance, [Vinciarelli and Luettin 2001] hypothesizes that, for deslanted text, the number of columns containing a continuous stroke is maximum. Many methods proposed, therefore, to perform a shear operation for each angle in a predetermined interval, and to measure the property a deslanted text would maximize in order to determine the slant angle. Authors usually differ in the property they propose to maximize. For instance, [Guillevic and Suen 1994] proposes to maximize the highest positive value of the first derivative of the vertical density histogram, [Pastor *et al.* 2004] propose to maximize the variance of the vertical projection which is quite similar to [Côté *et al.* 1998] where the lowest value of entropy is minimized. Before trying more sophisticated methods, we have applied the uniform deslanting technique proposed by [Pastor *et al.* 2004] although, after observing some badly deslanted images (especially short lines or isolated words), we have considered other techniques which ultimately has motivated the development of a new proposal including a non-uniform slant correction technique.

Figure 190: Best slant does not always put vertical longer strokes. Trying to put vertical the 'y' longer stroke is in conflict with the slant angle of 'p' grapheme. The same problem appears with other graphemes ('X', 'A', '7',...).

We believe that the methods described before have some drawbacks due to the heuristic nature they are based on. Slant is usually described as *the direction of vertical text strokes*, but this definition poses some problems since some long text strokes are not naturally vertical as illustrated in Figure 190. We have to be cautious since the actual use of this preprocessing is automatic recognition and not human judgment. Another problem is the difficulty to deslant lines with no vertical strokes. We believe that more powerful techniques based on shape classification techniques are more suitable in this stage. Machine learning algorithms can be used to determine whether or not a piece of handwritten material is slanted or not. These techniques can take into account the shape of strokes to detect those cases where strokes are not naturally vertical without performing grapheme classifications.



Relating the discussion on the actual use of deslanting, it is clear that we have chosen to follow the "human judgment" approach since we believe that this criterion improves the separation of graphemes into

different image columns and reduce the variability of outcomes from the same graphemes. Another advantages of this approach is the easy to ask humans to label data and the possibility of using the generated images to reduce the reading effort in the context of assisted transcription tools. The main drawbacks of this approach are twofold: some human effort is required and the criterion we are following is not directly related to the reduction of recognition errors. Other techniques based on machine learning which does not require human supervision, guided by WER, can be considered in the future.

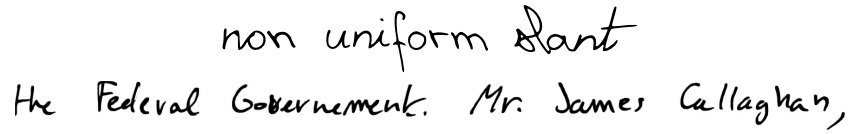


Figure 191: Handwritten line generated to illustrate the presence of non-uniform slant (up) and a real image from the IAMdb (down).

14.4.1 Non-uniform slant removal

Another important issue to consider is the presence of non-uniform slant, illustrated in Figure 191. This problem has been addressed in the literature in different ways. A simple methods to deal with different slant angles in the same line is to break the line into segments and to apply uniform slant correction on them. A more powerful technique to normalize non-uniform slant is proposed in [Uchida *et al.* 2001; Kuhn 2005] where a dynamic programming (DP) method is used to estimate the local slant angle for every image column in a way that two different criteria are maximized. On the one side, local slant angle estimates maximize some type of fitness measure (not surprisingly, based on geometrical heuristics), on the other side, the slant angle should satisfy a smoothness criterion (the slant angle must not change more than $\pm 1^{\circ}$ per column). We have also used dynamic programming and the same type of smoothness criterion, but we have used supervised learning techniques to determine whether or not an image column is slanted. To this purpose, a graphical tool (Figure 193) has been created and the 1000 text line images used to train the classification of interesting points have been manually deslanted after removing the slope.

The result of the deslanting supervised process is slant angle for every image column (obtained by linear interpolation of the control line angles). This data has been used to train a MLP (called Slant-MLP) to determine whether or not an image is correctly deslanted. Text line images are resized to height 40 for this purpose. Slant-MLP input is a 40×40 pixels window centered at the column the slant is computed. During training, training lines have been artificially sheared and feed to the Slant-MLP. The target output is a single value between 0 and 1 which reaches 1 when the shear applied to the training pattern matches the supervised slant angle and decays to 0 as the angle differs following a bell shape function. After some topology trials, the best Slant-MLP topology comprises two hidden layers with 64 and 32 neu-

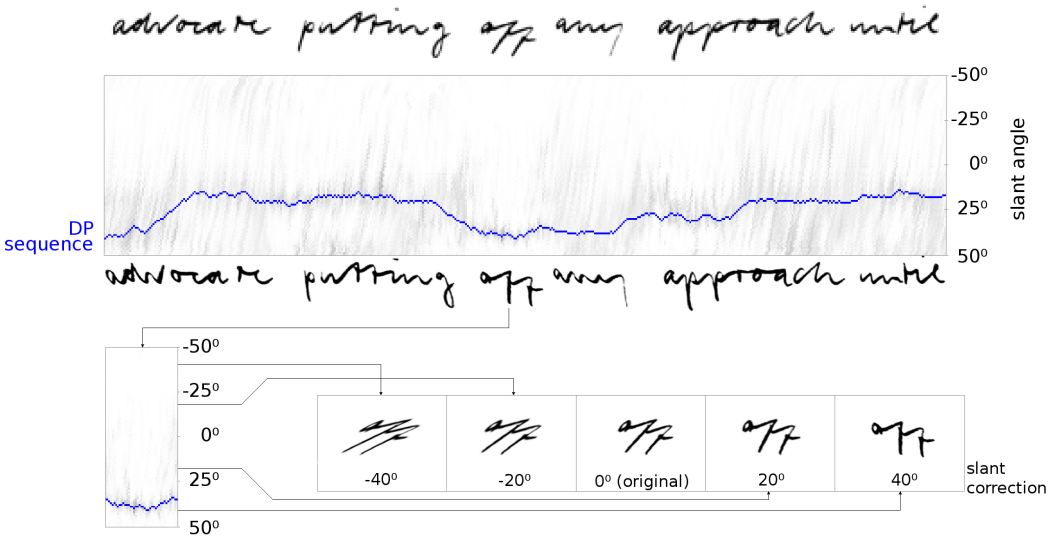


Figure 192: An example of slant removal (taken from [España-Boquera *et al.* 2011; Fig. 4]): the original text line image (top) and the slant-corrected text line image (bottom). The Slant-MLP estimates a measure of the slant angle for each column, shown as a gray level matrix. A dynamic programming algorithm obtains the optimal sequence of slant angles. Beneath the matrix is a detail of it for a segment of the text line image.

rons respectively and 1 output neuron. Once the Slant-MLP is trained, it is used to deslant as follows:

a) first, a global average slant angle is estimated and removed using [Pastor *et al.* 2004] as described in the previous section on uniform slant removal. This stage avoids to apply the technique on line segments since the method is less robust on short images and we only need to remove the average slant to reduce the angle interval of following steps. The globally deslanted image is resized to height 40 preserving the aspect ratio;

b) a shear is applied on previous image for each angle in a predetermined interval (in this case, [-50°,50°] as before). For every angle, a sliding window of size 40×40 traverses the sheared image and Slant-MLP is used to estimate how well deslanted (a continuous value) is that sheared image on every column. The result of applying Slant-MLP on every column and shear angle is a matrix of scores similar to [Kuhn 2005]. Figure 192 illustrates this process showing the matrix of scores and detailing how these values have been computed for a given column (around first 'f' of word 'off');

c) a dynamic programming algorithm is applied over this matrix to obtain the path with maximum addition of scores which also satisfies a smoothness criterion: the slant angle must not change more than $\pm 1°$ per column;

d) the final image is obtained by means of a non-uniform shear.

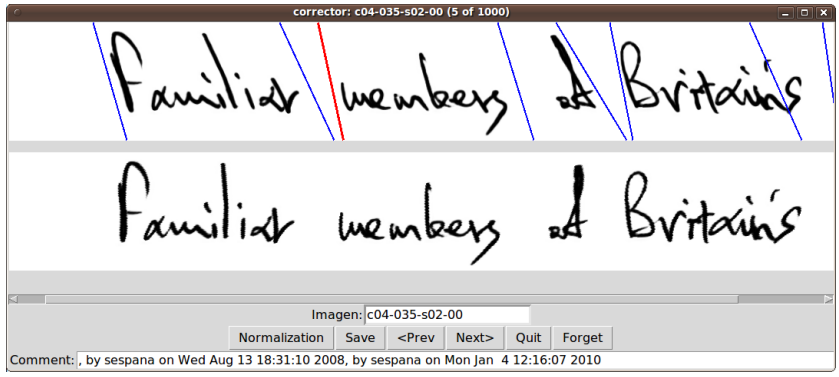


Figure 193: Screenshot of the graphical tool designed to correct the slant in a non-uniform way. Upper panel allows the user to insert, to remove or to modify control slant angles. Lower panel shows the image after applying non uniform slant correction where slant angles are obtained by linear interpolation of control line angles.

The whole slant removal process is illustrated in Figure 192 and an example of a slant-corrected image is shown in Figure 188(f).

14.5 SIZE NORMALIZATION

When the image is slope and slant-corrected, it is convenient to normalize the size of the text line in order to minimize the variations in size and position of its three zones (main body area, ascenders, and descenders). As with other preprocessing steps, we try to reduce the intra-class variability while preserving, at the same time, the inter-class variability, and yet some size variations help to differentiate between some characters like 'o' and 'O', 'P' and 'p' and so on. Fortunately, vertical alignment of baselines allows us to distinguish these characters. That's why we prefer to apply the size normalization stage at least at the line level[18] rather than at smaller scales as is also reported in the literature. For instance, in [Kozielski *et al.* 2012] frames obtained by means of a sliding window are independently normalized in size using moments, as illustrated in Figure 194.

We can observe that the importance of size normalization is very dependent on the feature extraction used afterwards since some feature extraction approaches are much more sensible to vertical translations. For instance, Section 14.6 reviews two main feature extraction methods: either based on geometrical characteristics or based on dividing the image column into cells (e.g. the raw pixels). The last method is usually less robust against vertical translations. It is also common to reduce the size of ascenders and descenders with respect to the main body area in this last approach, since they are not as informative (the presence or absence of ascenders and descenders is preserved, as

18 We are considering to generalize the preprocessing stages from line to paragraph/page level in order to take advantage of the correlation among lines if some a priori information allows us to guess that lines share common features such as size or slant (e.g. because they are written by the same author).

well as the width, but not the actual height), although the relevance of this compression is reduced when dimensionality reduction techniques (i.e. PCA, autoencoders,...) are applied and may also depend on the model using these features.[19]

Although size normalization could take into account height and width simultaneously, we will tackle each one separately, which does not imply that they are independent since, for instance, width normalization may take into account information obtained from height normalization. Let us remark that the effects of height and width on decoding are very different: height normalization is basically related to feature extraction and estimation of observed emissions in HMMs (in case these models are used as segment models) whereas width normalization is related to model duration which is addressed by means of the HMM topology (number of states, explicit duration models,...).

(a) Sample image slices. **(b)** Their size normalized versions.

Figure 194: Image taken from [Kozielski *et al.* 2012; Fig. 5] illustrating the possibility of performing size normalization locally on the slices obtained by the sliding window used to obtain the sequence of frames.

14.5.1 Vertical/Weight normalization

The aim of the weight normalization process is to produce an image of a fixed height no matter the original height of the deslanted image. Some works in the literature call this stage "vertical normalization" and a typical way to perform this normalization consist of estimating the baselines of the image line by means of the horizontal projection [Schlapbach and Bunke 2005; Gorbe-Moya 2005]. Our approach to size normalization is to detect the reference baselines by extracting and classifying interesting points and to normalize the size of the line according to them. As explained before, interesting points may be classified into three classes each type (depending on the type of local extrema detected, as illustrated in Figure 184) or into five classes if we ignore the type of local extrema.[20] We have used this last approach in order to train only one MLP (from now on called Normalize-MLP). Points belonging to the same class are used to obtain each reference line by linear interpolation (see Figure 188(g)).

Height normalization it is performed for each column of the image by linearly scaling the three zones (area of ascenders, main body area, area of descenders, see Figure 177) so that each area is placed in a different set of image rows (as is already described in the literature, see

19 For instance, MLPs are more robust than Gaussian mixtures with diagonal covariance matrices with against correlated and superfluous parameters.

20 In this case, we can use the type of local extrema to discard two outputs from the net. The comparison of each approach has not been performed yet but is planned as future work.

Figures 182 (d) and (e)). There exists several possibilities to perform this scaling:

- to linearly scale each zone to a fixed height column-wise. The main body area is usually assigned more height than ascenders and descenders as explained before. See Figure 188(h) for an example of an image normalized in this way;
- to scale the height of the main body area column-wise and to preserve the relative height of the area of ascenders and the area of descenders with respect the main body area. The area of ascenders and descenders of the height normalized line being fixed, some tall ascender or descender strokes could be cropped. The main aim of this approach is to avoid some artifacts caused by the expansion of very narrow ascender and descender zones and to preserve the vertical aspect ratio of each zone;
- to scale the height of the entire image by performing a sole image scaling and vertical translation of the entire image line in order to place the horizontal average of the upper baseline and the horizontal average of the lower baseline in the same position of the size normalized image. Since the normalized image has a fixed height, this option might partially crop some tall ascenders or descenders.

Relating the last approach, it seems a pity to apply a quite sophisticated approach for tracking reference lines just to extract two vertical reference lines. The rationale behind this approach is two obtain a more robust method which can be combined with previous ones.[21] It is expected that this less sophisticated preprocessing techniques are more tolerant facing misclassified interesting points. The proposed preprocessing is not perfect. Indeed, we have empirically observed that most mistakes are due to line segmentation mistakes,[22] which are propagated to this stage of the pipeline, as illustrated in Figure 195.

Unfortunately, the number of experimental validations has to be limited by pragmatic reasons and only the first option has been tried for the moment in this work (see Chapter 15): all lines are set to height 40, ascenders are reduced to 20% of the final image height and descenders are reduced to 10%. The height 40 has been chosen to set the height of the main body area roughly similar to the average of this area observed on the training set.

14.5.2 Width normalization

It should be noted that the proposed height normalization techniques do not preserve the aspect ratio, which is a source of variability

21 Although this alternative height normalization could be used to train an independent recognizer in order to combine it with others by means of techniques such as ROVER combination [Fiscus 1997], it is better to take profit that columns from each height normalization approach can be synchronized so that frames computed by each one may be used in parallel by a sole decoder.

22 Most experimentation described in this work has been performed on lines from the IAMdb which is a line segmented database where users were asked to copy some text in forms and to use a rule. Nevertheless, some incorrectly segmented lines exists in the database.

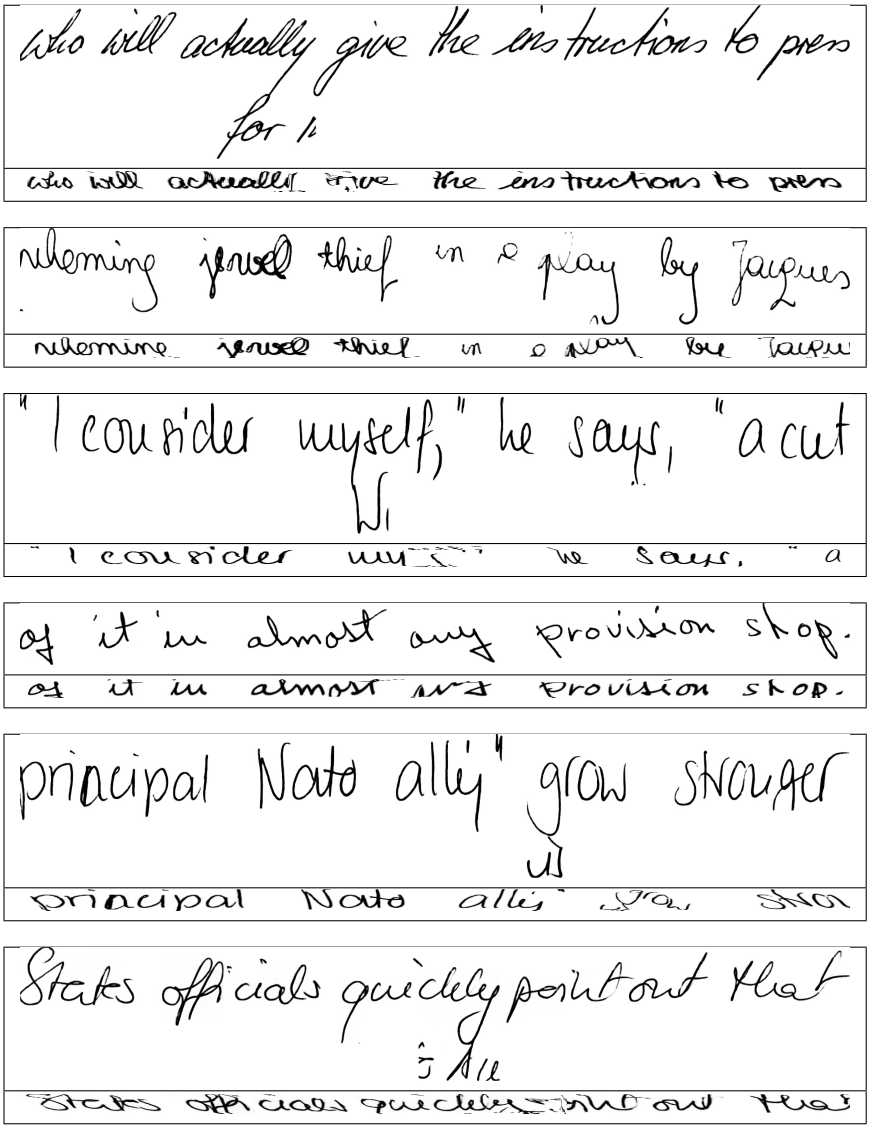


Figure 195: Some examples illustrating how our preprocessing may fail. Each example contains two images: the original line on the top and the same image line, after the preprocessing, on the bottom.

Figure 196: Example of two word images from the IAMdb preserving the original (relative) size (up) and the same images after preprocessing (down). As can be observed, normalizing the height while preserving width produces a greater width variability.

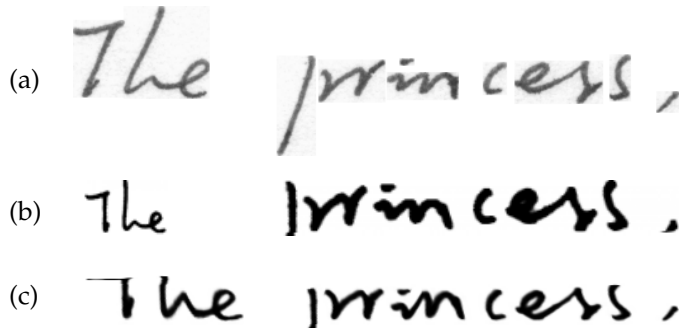


Figure 197: Comparison of two different image normalization methods. From top to bottom: (a) original image extracted from the IAMdb, (b) image normalized with "second maxima" technique described in [Romero 2005], and (c) image normalized with our method.

where word images of different sizes (from the same database) are scaled to a fixed height while preserving the width, as shown in Figure 196. This variability seems similar to the speech speed variability and can be taken into account, at some extent, by HMMs or by other segment estimation techniques. The aim of width normalization is to reduced or to alleviate this problem by scaling the image width so that the average character length is roughly the same for all lines.

We will assume that writers do not drastically change the size of graphemes within the same line meaning that a global horizontal scaling over the height normalized image would suffice, but other systems described in the literature propose to perform a size normalization (including width normalization) at lower scales. For instance, [Romero 2005] proposes to divide lines into segments (in the same way as done with slope and slant correction) and to normalize the size of each segment (preserving the aspect ratio) independently. Our choice to ignore the size variability in the same line for width normalization purposes is less flexible but seems more robust facing bad classification of image segments, as shown in Figure 197. Not also that size normalization errors increase stroke thickness variability.[23] Despite applying a simple global horizontal scaling, several approaches may be considered:

23 Stroke thickness normalization has been avoided in this work.

- [Schlapbach and Bunke 2005; Liwicki *et al.* 2006b] propose a width normalization based on the number of black to white horizontal transitions. Although the description is not very detailed, we guess that this value is measured on the rows of the main body area. The ratio between this value and the image width is measured on the training set to obtain a reference which is used to obtain the horizontal scaling factor applied to a new image line;

- our technique to track reference lines more accurately allows us to measure the height of the main body area very accurately. It is then possible to measure the average vertical scaling applied to the main body area during height normalization in order to apply the same ratio horizontally. This approach to width normalization nearly preserves the aspect ratio;

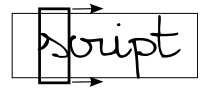
- the combination of both techniques (both ratios could be averaged) and some extensions in order to obtain a non-uniform width normalization (by estimating these values locally but then applying some restrictions on the change of the ratio, as was also done in our non-uniform slant correction by means of DP) can also be considered as future work.

Before trying more sophisticated width normalization approaches, we decided to work with height normalized images without performing any kind of width normalization in a first stage, obtaining quite good results. It is clear that the proposed width normalization techniques have to be tried in order to analyze their effect and even used to obtain different preprocessed images to train not completely correlated recognizers to be combined afterwards. Nevertheless, since the combination of several possibilities at each preprocessing stage leads to a combinatorial explosion, we have planned to postpone experimentation with width normalization to future work.

14.6 FEATURE EXTRACTION

After preprocessing, a feature extraction method is applied to capture the most relevant characteristics of the character to recognize. Preprocessing, feature extraction and classification methods are quite dependent. We are thus interested in feature extraction methods suitable for being used by HMMs with our proposed preprocessing. In particular, some feature extraction techniques specially devoted to isolated character recognition or to holistic classifiers[24] are not covered here.

Most feature extraction methods used by HMMs are based on sliding windows as is also done in speech. The sliding window captures an image of fixed width and is moved in the reading order to obtain a sequence of feature vectors. They usually differ in the features



24 The use of holistic classifiers is discussed in this work as a complementary improvement of the proposed techniques (either in the general problem of joint segmentation and classification in Section 7.7, or in the particular case of handwriting in Section 15.5). These classifiers may take advantage of global features, such as the presence of loops.

extracted from these slices, which can be roughly divided into two groups (both type of features can be used jointly):

- geometrical features;
- divide the slice cells. A particular case would consist on just using the raw pixels as features, as proposed by some authors.

Geometrical features

The most notable example of these types of features is proposed in [Marti and Bunke 2000] where nine geometrical characteristics[25] are obtained from a 1 column width (avoiding overlap between consecutive window positions) sliding window:

- number of black pixels, center of gravity, the second order momentum of the window;
- position of the upper and lower contour in the window;
- orientation of the upper and the lower contour in the window (gradient of the contour at window's position);
- number of black-white transitions in vertical direction;
- number of black pixels between the upper and lower contour.

Zoning/cells features

Other works propose to divide each image column into cells and to extract some features from each cell. For instance, [Bazzi *et al.* 1999] use 20 equal overlapping cells (columns may be overlapped, cells also, overlapping factor is a parameter) in order to extract the following features from them:

- intensity (percentage of black pixels within each cell) as a function of vertical position;
- vertical derivative of intensity (across vertical cells);
- horizontal derivative of intensity (across overlapping frames);
- local slope and correlation across a window of two cells square.

Based on similar ideas of [Bazzi *et al.* 1999], [Toselli *et al.* 2004] also uses an overlapped sliding window to divide the image into cells in order to extract the following three parameters from each cell (as illustrated in Figure 198):

- normalized gray level, which is obtained by smoothing the normalized image with a 2-d Gaussian filter;
- horizontal gray level derivative, which is computed as the slope of the line which best fits the horizontal function of column-averaged gray levels; and
- vertical gray level derivative, which is computed in a similar way.

25 Notice that most of these features are described for binary images.

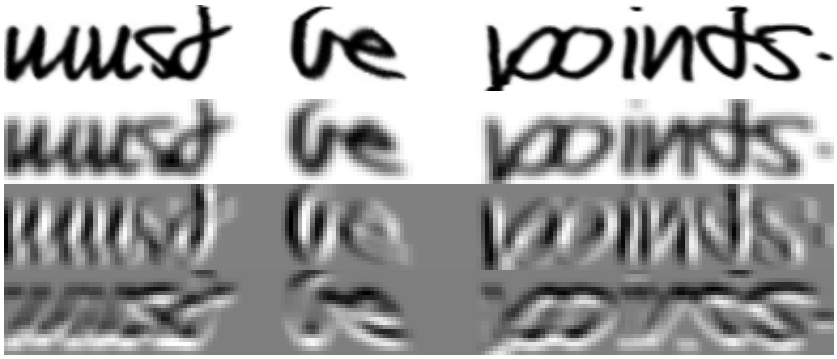


Figure 198: Preprocessed image (top) and features extracted using [Toselli *et al.* 2004] (bottom).

It is also possible to combine both types of features. For example, [Brakensiek *et al.* 2000] describe a system using 8 parameters corresponding to Discrete Cosine Transform (DCT) coefficients of a vertical column of pixels and 3 geometrical characteristics (height over baseline, thickness at the current horizontal position, number of black-white transitions). In [Kozielski *et al.* 2012] the raw pixels of the slice (after an independent size normalization illustrated in Figure 194 are used together with some complementary information obtained from statistical moments measured on the original images.

Each frame corresponds, ideally, to a sole grapheme, and the decoder expects that. Slant correction helps to achieve this important property, but even a slant corrected image may contain strokes (usually associated to ascenders or descenders) which spans neighboring characters. To address this problem, [Vinciarelli 2003] proposes a heuristic to remove parts of those problematic strokes, as shown in Figure 199. Unfortunately, this technique is not suitable for languages containing accents or graphemes with several connected components such as 'ñ'.

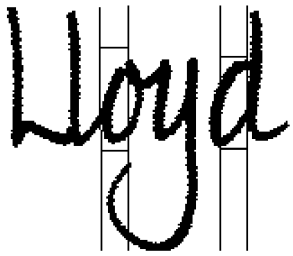


Figure 199: Image from [Vinciarelli 2003; Fig. 3.6] showing a method to remove ascender and descender strokes which probably belong to other graphemes: foreground pixels outside the core region not connected to the content of the core region (in the example, descender from 'y' behind 'o') are not taken into account in the feature extraction.

Note also that some features are more robust than others against some preprocessing mistakes. For instance, the derivative of the vertical contours proposed in [Marti and Bunke 2000] is less altered by vertical shifts than methods based on dividing the column into cells.

We have used the feature extraction method proposed by [Toselli *et al.* 2004] using 20 cells, leading to feature vector composed of 60 values. Since our normalized images had a fixed height of 40 pixels, each frame corresponds exactly to two pixel columns. Although it is also possible to apply dimensionality reduction and de-correlation

techniques to any of these parameters, we have not tried these techniques, which may affect some classifiers more than others.[26] All the features used in this work have been mean subtracted and divided by standard deviation estimated from training data.

14.6.1 Avoiding the feature extraction process

It is also possible to simply avoid the feature extraction process and use the raw pixels from the preprocessed text line image instead. In this case, we can take advantage of the fact that our preprocessed text line images have a fixed height so that a slidding window traversing this image can be used to feed a (possibly deep) MLP.[27] It seems even better to use convolutional neural networks[28] (CNNs). CNNs [LeCun *et al.* 1998] are organized as a series of convolutions (with kernel matrices) and pooling operations. Convolutions are applied on different kernels to obtain different transformations leading to different maps. They are often used to extract local features invariant to translation. On the other side, pooling operations may be applied to the maps produced by convolution layers in order to reduce the dimensionality. After one or several convolution and (optionally) max-pooling operations, a hidden layer contains a set of features that can be processed by a MLP.

We can observe that just one convolution layer suffices to obtain similar features as the feature extraction proposed by [Toselli *et al.* 2004] and extensively used in this work. We have not used CNNs in former experiments since CNNs were not been implemented until recently in our April toolkit, but the very preliminary results are highly promising and have clearly surpassed previous ones that used the aforementioned parametrization described in [Toselli *et al.* 2004].

14.7 EXTERNAL WORD OVER-SEGMENTATION

character segmentation is bad idea

Current state of the art techniques for HTR recognition are based on HMMs or other *segmentation free* approaches (see Chapter 4 for a description of other segmentation free techniques such as RNNs with CTC decoding). In particular, none of the works found in the literature[29] which tried to segment line images at the character level prior to their recognition has been successful. This idea is very well expressed in [Kaltenmeier *et al.* 1993]:

> *In cursive script the only a priori segmentable entities are whole words and not single characters. Hence, finding the right segmentation of a word is strongly related to the intrinsic classification*

26 In particular, HMM with GMMs based on diagonal covariance matrices does not perform so well with features as proposed in [Toselli *et al.* 2004] and can be highly improved by means of PCA or similar techniques.

27 We can also envisage a (handwritten) document recognition system lacking not only feature extraction but even the text line preprocessing. This system could be based on multidimensional ANNs [Graves *et al.* 2007]. Although we have some ideas in this line, this work would depart from the line of this work based on TSGMs.

28 Briefly mentioned in Section 4.4.4.

29 Several publications in this line are cited in [Plötz and Fink 2009].

> *task to be solved. Traditional "segmentation first" approaches might hardly be used for cursive script recognition. Some experiments were made with different approaches; none of them was really successful enough. Often, even for human readers, character segmentation can only be derived together with recognizing what was written, i.e. segmentation is performed implicitly during recognition. Technical systems should follow the same approach.*

which basically discourages us from trying to segment handwritten lines into characters by using "external segmentation" methods. By "external segmentation" we mean the opposite of "internal segmentation" where the segmentation is obtained as a by-product of a joint classification and segmentation process.[30] An alternative nomenclature for external and internal segmentation is, respectively, "segmentation first" and "segmentation free".

internal vs external

Although speech and handwriting show coarticulation effects at the phone and at the grapheme level, respectively, the case of handwriting has a particularity that we claim it should be exploited when available: while continuous speech usually lacks pauses between words and usually shows liason and across-word coarticulation phenomena, handwriting makes use of spaces to separate words, at least in most contemporary occidental handwriting systems. But the reasonable speed of HMM decoders, initially designed for ASR tasks, has made most researchers on HTR to ignore the possibilities of this difference.

spaces usually separate words

In this work we try to claim the possibility of using external over-segmentation in HMMs. We have discussed the use of external over-segmenters to reduce the search space in Sections 11.1.4 and 12.4. Now we provide some techniques we have implemented to over-segment handwritten lines at the word level. These technique will be used in Chapter 15 to compare, in terms of the decoding speed vs recognition accuracy, the over-segmenter proposed in this section with a classical decoding and with an "oracle" segmentation which is obtained with a forced alignment and can be considered a lower bound to this type of over-segmenters.

Related work

We have found very few works on text segmentation at the word level. For example, [Zimmermann and Bunke 2002a] try to automatically obtain the bounding boxes of word images in the IAMdb in a supervised manner in order to label the database. Therefore, this work is restricted to forced alignment and cannot be used for decoding. Most word segmentation techniques found in the literature [Seni and Cohen 1994; Mahadevan and Nagabushnam 1995; Manmatha and Rothfeder 2005; Huang and Srihari 2008; Papavassiliou *et al.* 2010] work as follows: first, the handwritten line is decomposed into connected components (CCs) or, sometimes, sets of horizontally overlapping CCs. The distances between adjacent components are computed following a gap metric. Finally, gaps are classified into "within" and "between" words. Usually, a simple threshold is obtained from the training data to perform this classification afterwards.

30 For example, by tracing back the best path in a HMM Viterbi decoder

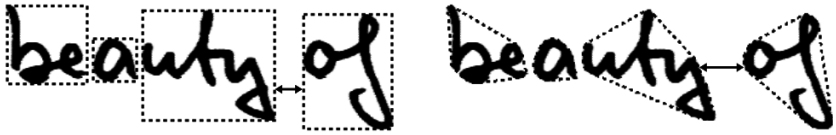


Figure 200: Example taken from [Varga 2006; Fig 5.2] illustrating components of a handwritten text and their bounding boxes (left) or convex hulls (right).

[Varga 2006] proposes a method which does not use a global threshold to classify gaps but which takes into account the context of these gaps, i.e. the relative sizes of the surrounding gaps. This method resembles a hierarchical top-down clustering algorithm and constructs a tree in a top down manner. Since punctuation marks may affect accuracy, a punctuation detection algorithm is used and the components classified as punctuation mark are not taken into account.

Two popular types of gap metric are the horizontal distance between the bounding boxes of components and the minimal horizontal distance between convex hulls of adjacent components, as illustrated in Figure 200. More recently, [Papavassiliou *et al.* 2010] introduced a novel gap measure which is more tolerant to some particular CCs shapes and performed an extensive comparison of different methods.

Proposed method

Note that the above methods tried to segment a line into words by classifying gaps separating CCs. Hence, they are solving a much harder problem than us since we are *over*segmenting. Moreover, we will take profit of a richer type of over-segmentation information proposed in this work (see Section 3.1.4). Usual over-segmenters have to choose between "this frame boundary do not separate words" or "this frame boundary may separate words", we can also say things like "there is a word boundary in any of these frame boundaries" which can be used by our decoders.

The proposed method works exclusively with horizontal projections of CCs, since it is very unlikely that strokes from different words are touching. We have not considered vertical projections to detect gaps, it is more likely to find strokes overlapped horizontally (see Figure 201). Our external over-segmenter proceeds as follows:

1. For every 8-connected component:[31]
    a) obtain the leftmost and the rightmost horizontal coordinates;
    b) mark as a possible word segment the frame frontier preceding the frame associated to the leftmost horizontal coordinate and the frame frontier following the frame associated to the rightmost horizontal coordinate;

31 There are several methods to detect and label connected components. For instance, see `http://en.wikipedia.org/wiki/Connected_Component_Labeling` or [Shapiro and Stockman 2002; Chapter 3]. We do not need to label the connected components but only to detect the leftmost and rightmost horizontal coordinates of every connected components. We have used a simple Depth First Search (DFS) using the pixels to temporarily store visited information. The two pass algorithm could perform slightly better, but speed here is negligible.

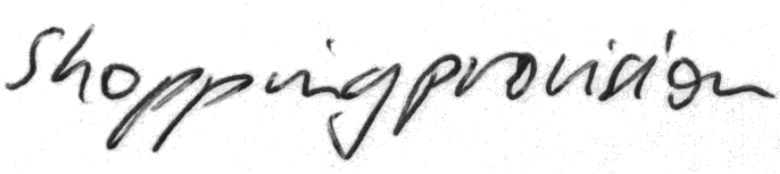


Figure 201: Image extracted from the IAMdb page `h07-025a` showing a space between words that a simple vertical histogram projection could not detect (at least without the proper slant correction). The words in this example, thought, do not share connected components.

c) mark frames between boundaries as "containing stroke".

2. detect consecutive frame sequences not marked as "containing stroke" and mark as "sure word boundary" those whose width is greater than a given threshold.[32]

14.8 ONLINE PREPROCESSING

Online HTR is devoted to handwriting acquired with special devices which capture the trajectory of the stylus while the user is writing. Since online can be trivially converted to offline, it is expected to be at least no more difficult than its offline counterpart. Indeed, online contains more information relating the number, order, direction and speed of strokes. For instance, overlapped characters pose a problem in offline but are easily recognized in the online case because they are not written simultaneously. Not surprisingly, online is usually considered easier than offline. On the other side, the same set of stroke images can be obtained by moving the stylus in many different ways and also the set of point coordinates can greatly vary depending on the writing speed. Our work in online HTR is much more modest than our work in offline since it has not been our focus of research. Historically, we have worked with offline data before and our interest in online HTR was been very limited: some experiences on bimodal recognition described in Section 15.6.

The main interest of this work is to illustrate the possibility of adapting the proposed offline preprocessing to the online counterpart. We have implemented a first version of this proposal by improving a quite standard online preprocessing chain obtaining better recognition results in the bimodal experiments reported in the aforementioned Section. The proposed online preprocessing can be improved as discussed below.

14.8.1 Online preprocessing stages

The online input data is basically a sequence of strokes, and a stroke, in turn, is a sequence of coordinates ordered in time usually accompanied with timing information (x_i, y_i, t_i). Visible or pen-down strokes

32 Note that, in order to avoid false positives, a large threshold has to be used so that the effective pruning due to these sure word frontiers is quite small.

correspond to the points capture while the stylus is touching the pad surface, whereas pen-up strokes are acquired when the stylus is not touching it. We are only interested in visible strokes since the others are non-informative to perform HTR. Moreover, no stylus pressure information corresponding to writing strokes is taken into account.

A basic preprocessing can be on a stroke-by-stroke basis and involves the following steps which preserve the temporal order of the data points:

Removing duplicated points

Repeated or duplicated points appear in a trajectory when the stylus remains motionless for some time. They can be removed by checking whether the coordinates are the same as the previous point.

Elimination of Hooks

Usually, hooks occur at the beginning and/or at the end of a stroke. [Huang *et al.* 2009] describes a method to remove them. We do not apply this step, but it is cited here for the sake of completeness.

Noise reduction or smoothing

Handwritten strokes may suffer from erratic hand motions and inaccuracy of the digitalization process. Several smoothing techniques have been proposed to reduce this noise, among them, the easiest method probably consists of replacing every point in the trajectory by the mean value of its neighbors [Jaeger *et al.* 2000], although more sophisticated method like one based on cubic splines [Huang *et al.* 2009] can be found. We have chosen the first method.

Size normalization

Has the same purpose of the offline counterpart. Usually affine transforms translation and scaling are performed to this end. Next section will detail this step.

Resampling and speed normalization

Equal-length trace segmentation or equidistant resampling of the point sequence is a technique that can be used to normalize the writing speed consisting in resampling every stroke using a fixed distance interval. We have followed this technique which has been used in many other works [Liwicki *et al.* 2006a; Huang *et al.* 2009].

This method is controlled by a parameter called resampling distance. The need of a tuning parameter is a drawback: the amount of data points decreases with the resampling distance, which is good to reduce the computational recognition effort but might lose information. On the contrary, if the value of the resampling distance is too small, the size of the sequence to be decoded is increased while the amount of relevant information remains the same.

Another method called "derivatives normalization" is proposed in [Pastor *et al.* 2005] to circumvent trace segmentation while achieving

invariance to speed by normalizing the derivatives by the derivative module in each point.

14.8.2 Adapting the proposed offline preprocessing

The purpose of the adaptation from offline to online is to take profit of the offline preprocessing ideas described in previous sections as well as the implementation and training effort performed on IAMdb which makes sense in our case since we have used this techniques with a visually similar online corpus [Liwicki and Bunke 2005] (which has been used, in turn, to create a bimodal corpora [Pastor *et al.* 2009]).

After the repeated point elimination and the smoothing steps, the online line is converted to an image[33] which goes through the same process as the offline images (obviously avoiding image cleaning): the line image is divided into segments to correct the slope, the slant and to obtain the text baselines. We have not performed non-uniform slant correction in this case. With this information, we try to normalize the original online data. Instead of converting the offline data back to online, as described in [Lallican *et al.* 2000], we simply keep a trace of all transformations applied to the image during the preprocessing in order to apply the same set of affine transforms to every stroke, as depicted in Figure 202. Note that dividing the line into blocks never breaks strokes and we have avoided non-uniform slant correction. In order to rely just on affine transformations, size normalization is performed by means of a the global scaling after estimating the horizontal baselines of the main body area from the more accurate reference lines obtained by classifying interesting points. The resulting online sequence is desloped, deslanted and normalized in size by means of a scaling and a translation so that the lower and upper baselines are located in $y = 0$ and $y = 1$ respectively.

14.8.3 Feature Extraction

The set of features for online HTR that we can find in the literature can be divided into two classes [Liwicki *et al.* 2006a]: On the one side, features extracted for each stroke point, considering its neighbors. On the other, features related to the relationship of this point in the off-line version, usually with pixels surrounding the point. We have only used a subset of 8 classical features related to the stroke points:

- vertical y coordinate;
- the absolute horizontal coordinate x is **not** used since this value would depend on the position of the grapheme in the line. Other authors [Liwicki *et al.* 2006a] use this value after applying a high-pass filtering which subtracts a moving average;
- first and second derivatives of vertical and horizontal positions;
- curvature or the inverse of the radius of the curve in each point;
- speed: the module of the velocity. Original timing information is kept during trace resampling in order to compute this value;

33 Generating a `svg` file and using `inkscape.org` to obtain the bitmap grayscale image.

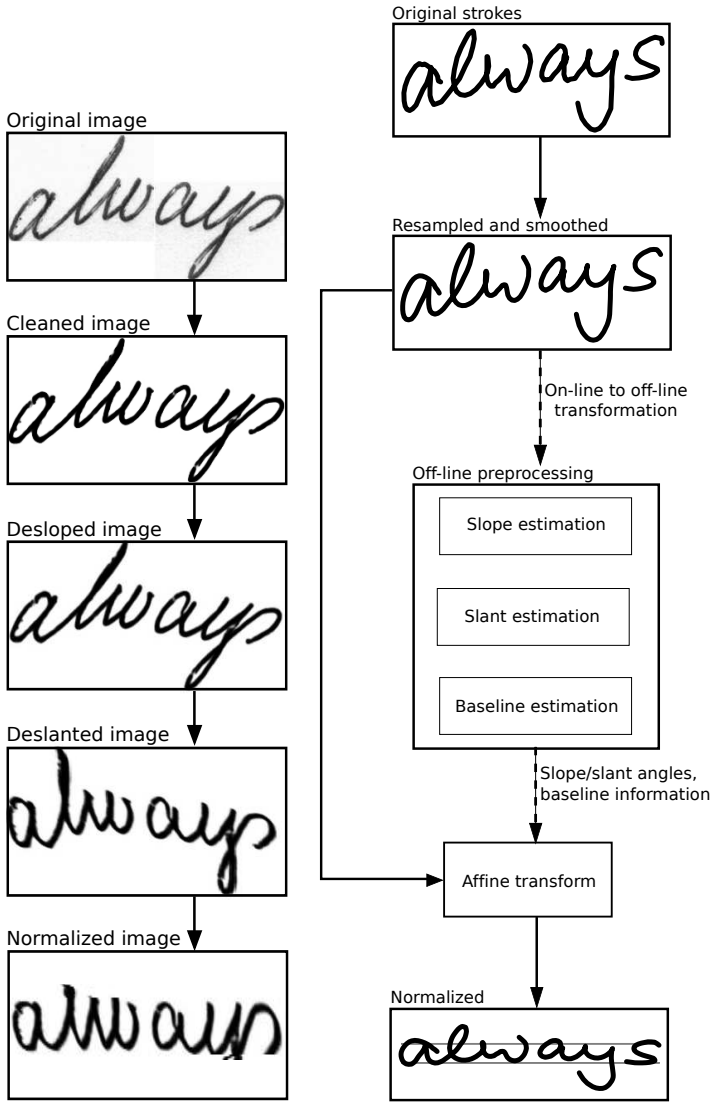


Figure 202: Preprocessing example of an offline sample (left), and an online sample (right). Image taken from [España-Boquera *et al.* 2010; Fig. 3].

- a boolean value which marks the end of each stroke. [Liwicki *et al.* 2006a] prefer to include a pen-up/pen-down value and to connect consecutive strokes with straight lines.

14.8.4 External word over-segmentation

We know that all points from the same stroke belong to the same connected component and we have already included a boolean parameter to mark the last point of every stroke. In this way, it is very easy to add over-segmentation marks in frame boundaries separating strokes. We should determine if it is reasonable to assume that writers *never* join different words with the same stroke. A naive adaptation of the offline over-segmenter, without converting the online data to an actual image, would only need to determine which set of (consecutive) strokes are touching to consider them belonging to the same CC.

14.9 SUMMARY AND SOME CONCLUSIONS

Several preprocessing techniques for offline and online handwritten text have been proposed in this chapter as well as the description of most representative alternative techniques found in the literature.

The constributions of this chapter follow the ideas described in Section 2.9.2 (graphically depicted in Figure 20) relating the use of learning techniques from the very early stages of recognition and the idea of delaying hard decisions as much as possible in order to have more higher level information available.[34] Relating the use of machine learning techniques for document analysis and recognition, we can highlight the use of connectionist methods which are reviewed in [Marinai *et al.* 2005]. Indeed, our techniques for offline text recognition, introduced in [Gorbe-Moya *et al.* 2008; España-Boquera *et al.* 2011], are vastly based on ANNs to perform several tasks ranging from image cleaning and enhancement to non-uniform slant correction. Moreover, they are not limited to preprocessing but also to classification (either as estimators of emission probabilities in hybrid HMM/ANN models, or as holistic classifiers for short words, as described in the following chapter).

vastly based on ANNs

We believe that the benefits of this preprocessing is not limited to improve HTR tasks but also to facilitate the readability of handwritten lines[35] which can be used, for instance, to reduce the human effort in interactive transcription scenarios. This feature has not been empirically validated yet. On the other side, their validation in HTR tasks, with the IAMdb corpus, is delayed to the next chapter.

other applications

We have also adapted opur proposal from offline to online data by converting the stroke trajectories into images which are manipulated as offline images *while keeping and applying at any moment the same trans-*

34 As future work we could improve the delaying of hard decisions as well as take into account other ideas such as using information from later recognition stages to improve the first stages, they are detailed below but are not taken into account for the moment.

35 In this case, choosing a height normalization approach which avoids the compression of ascenders and descenders.

formations to the online counterpart as described in [España-Boquera *et al.* 2010]. Let us remark that better ways to perform online preprocessing are possible using the same ideas instead of just relying on the offline version. Also, the transformations applied to the online version are restricted to affine transforms, so we have avoided in this first stage non-uniform slant removal and a more accurate height normalization.

critics

Needless to say, the techniques described in this work can be further improved and are not exempt from critics. Relating the critics, one of the most obvious one is the somewhat arbitrary choice of parameters in some stages. For instance: why the non-uniform slant removal tries a range of angles from -50° to 50° one angle at a time? One of the reasons is that, in many cases, we have been biased by what is reported in other works from the literature. This is not a complete justification but seems a reasonable criterion when more exhaustive trials would lead to an non affordable combinatorial explosion. In most other cases, preliminary informal trials as well as previous experiments have helped us to reduce the set of parameter configurations. In other cases, some parameters are not even considered because they would lead to too large computation times. Finally, some values are extracted from data: the line height of normalized images is obtained from the median of heights in the training corpus.

choice of parameters

computational cost

Another drawback is the expensive computational cost of some preprocessing stages relating other techniques, especially the classification of "interesting points" where a large context of the image around each point has to be compressed using a fish-eye distortion and classified by a MLP. Another costly operation, compared with more traditional approaches, is the non-uniform slant removal which seems an example of the law of diminishing returns mentioned in Chapter 1 since most lines from the corpus we have tried show a uniform slant. Nevertheless, we are quite confident about the possibility to significantly reduce the cost of many stages without sacrificing performance.

Some preprocessing stages are only expensive in relative terms. For instance, the use of ANNs to perform image cleaning and enhancement is, no doubt, computationally more expensive than most heuristic binarization methods. But this cost is negligible compared with the overall cost of preprocessing and decoding. The use of image enhancement techniques based on supervised learning techniques, as proposed in this work, will become more and more important when migrating from the relatively clean images of IAMdb to ancient, degraded or simply with more realistic documents. There are some recent and related works in this line [Pastor-Pellicer *et al.* 2013; 2015c;a]

limited human effort

There is a critic which deserves special attention: since the process is not fully automatic but semi-supervised, human effort is required every time the corpus or the task is changed. The overall human effort required in this process for the IAMdb, including the fact that the technique itself was developed at the same time, and using a quite naive graphical application (illustrated in Figure 186) to apply the bootstrapping process, can be measured in around 30 hours (excepting the non-uniform slant correction which was performed later by means of another graphical tool, shown in Figure 193, which required about the same time as the previous stage). We can consider that the ratio between the applied human effort and the obtained benefit is worth the

effort, not to mention the fact that this effort will be surely reduced in the future:

- nowadays, a much powerful application is available which integrates the stages of previous graphical tools depicted in Figures 186 and 193 in a more intuitive and agile way [Martínez-Morató 2011]. It is expected to integrate this graphical tool in an interactive transcription system in order to be able to allow the end user to correct the preprocessing when required so that these corrections, as well as the corrections on the transcriptions, could be used to retrain the models;
- since this labeling effort will be associated most of the time with the migration to other tasks, we have empirically observed that models trained with one corpus perform quite well in a different one provided they are not much different. In particular, models trained for the IAMdb have been successfully used *without modification* in the french RIMES database, as described in [Zamora-Martínez *et al.* 2010], with good results (indeed, we have achieved the second position in ICDAR'09 handwriting competition [Grosicki and El Abed 2009] using this method);
- since the transcription of handwritten lines is often at our disposal, it can be used to improve the bootstrapping process in a semi-supervised approach by selecting first the set of lines which are not well recognized and even to avoid the human validation ond those which were recognized without mistakes;
- it is also possible to automatically try different labels in the classification of some interesting points or to vary the angles of the non-uniform slant removal process by trying different models or by perturbing the outputs in order to have several choices at our disposal. It is then possible to assigning a confidence to these labels in terms of the overall performance of the resulting parameters by using a decoder to check which ones perform better. In this way, besides filtering some samples from the database to reduce the labeling effort, it is also possible to label new patterns in a semi-supervised way.

improvements

Relating the possible improvements on the proposed preprocessing, it is clear that each independent stage can be improved and also that more information should be taken into account when several lines from the same writer are preprocessed instead of applying the preprocessing individually to each line. In particular:

- the detection of interesting points can be improved in order to reduce the number of points and also to avoid the explicit computation of horizontal contours, specially in degraded documents, where the detection of these contours is not as reliable;
- it is possible to study the relevance of using three different classifiers to label interesting points: one for slope removal and two for size normalization. Nowadays the size normalization is based on a unique MLP with 5 outputs instead of two MLPs with 3 outputs each one, one for each different type of "natural" interesting point, as explained through the chapter;

- we also believe that some interesting points are more easy to classify by means of computationally cheaper methods which can also detect difficult cases in order to rely on more accurate techniques. This would lead to a pre-classifier which would ameliorate the computational performance of the process while limiting the loss in quality;
- it is mandatory to make the overall methods more robust and tolerant to variations in the image resolution (dpi's) and in the size when migrating from research to exploitation stage;
- it would be very interesting to study if it is possible to adapt these techniques to other writing systems such as arabic handwriting;

using raw pixels and CNNs

Besides enhancing the preprocessing, we can also tackle the feature extraction: preliminary experiments have shown that the overall recognition system can be greatly improved when replacing the current feature extraction process based on [Toselli *et al.* 2004] by the use of the raw preprocessed image to feed CNNs, they are briefly described in the next chapter.

extensions

Finally, to conclude this chapter, we can also enumerate some further extensions (although the list is far from exhaustive):

- since most preprocessing stages require taking hard decisions from soft estimations, it is possible to produce not just one result but a set of results which can be represented in a compact way as discussed in Section 2.9.3;
- a modified version of the preprocessing stage, which includes perturbation models in the line of [Varga and Bunke 2003], is planned to generate several versions of each sample to increase the size of the training corpus. In particular, our preprocessing extracts lot of information which can be used to apply perturbation techniques more sophisticated than those proposed in [Varga 2006] (this work has been an inspiration for this idea). The applicability of perturbed versions of preprocessed lines is not restricted to improve the training process but it could be also used during recognition by simulating an ensemble and combining the results afterwards. Nevertheless, we believe that it is simpler to actually try several models with different preprocessing approaches and features to exploit the ensemble idea;
- it is obvious that the proposed techniques should be extended from text lines to whole pages to perform text line segmentation and extraction or even more complex layout analysis processes.

15 HTR EXPERIMENTATION

You may not be able to read
a doctor's handwriting and prescription,
but you'll notice
his bills are neatly typewritten.

Earl Wilson

CONTENTS

Unconstrained handwritten text recognition (HTR) of text line images is one of the most notable examples of the joint segmentation and classification on sequences: after layout analysis and text line extraction have been applied, the result can be considered as a one-dimensional sequence where frames can be obtained by chopping the line into small pieces in the order of the lecture (as depicted in Figure 1). Indeed, the Sayre's paradox [Sayre 1973] has been defined for this task.[1] The purpose of this chapter is to empirically validate the preprocessing techniques described in Chapter 14 as well as to apply some of the decoding techniques described in this work.

Although we have also successfully applied our decoders in ASR tasks (for instance using the French dialogue Media corpus achieving competitive results [Zamora-Martinez *et al.* 2012]), we have finally opted for limiting the exposition of the experimental work to HTR where different input modalities have been used (offline, but also online-offline bimodal). Most of the results of this chapter are also reported elsewhere (references will be properly cited). Experiments show the historical evolution of proposed techniques, of our April toolkit and of our own know-how, which means that some of the first results would be done right now in different ways and are overcome by newest results.

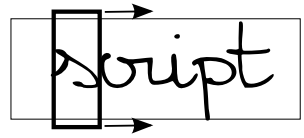
1 At least for word recognition in opposition to isolated or discrete character recognition, see Figure 11.

In order to properly compare recognition results, it is important not only to assure that corpora and training, validation and test partitions are the same but also that LMs, lexicons and other parameters coincide. Assessment measure criteria is also important (e.g. the use of case sensitiveness or taking punctuation marks into account when measuring Word Error Rate (WER)). The best thing that can be done to this respect, is to clearly report experimental conditions and, if possible, to share components and criteria with other authors. Some caution must be taken when comparing results with other authors as will be done several times through this chapter. We will try to clearly state the known discrepancies in order to reduce the risk of misunderstandings.

Chapter structure

- We will start with some experiments comparing our preprocessing with a more classical one, by using standard HMM/GMMs and count-based n-gram LMs. Next, some experiments with HMM/ANNs are reported. The purpose of this experimentation is to measure the suitability of the proposed preprocessing, the use of hybrid models in unconstrained HTR and to evaluate the influence of some parameters such as the dictionary size. This part basically corresponds to [España-Boquera *et al.* 2011].
- Experiments with NN LMs (described in Sections 6.6.1 and 9.5) are reported, they are compared with BLSTMs with CTC decoding (see Section 4.4.3) and both systems are combined leading to improved results. These experiments correspond to [Zamora-Martínez *et al.* 2014].
- Some preliminary results replacing the feature extraction used so far (based on [Toselli *et al.* 2004]) by CNNs are described. Although they have been performed in collaboration with another member of our research group, they are mostly part of his own PhD. Nevertheless, we consider convenient to discuss them because they benefit from the same preprocessing and decoders of this work. As with some connectionist LM experiments, they can be considered as a collaborative work that can provide indirect evidences of our work.
- The problems of OOV words has been discussed in this work and the use of sub-word LMs has been proposed. Some experiments related to lexicon-free recognition based on [Zamora-Martínez *et al.* 2010b] has been reported in Section 15.4.
- Some preliminary works relating the use of holistic classifiers [Zamora-Martínez *et al.* 2010a] are addressed in Section 15.5.
- Some experiments with bimodal (offline + online) are reported in Section 15.6. They were performed to participate in a contest. The interest of this work (results basically correspond to [España-Boquera *et al.* 2010]) is to empirically validate the adaptation of the preprocessing ideas proposed in this work from the offline to the online modality as described in Section 14.8.1.
- Finally, some conclusions and future work close the chapter.

15.1 HTR WITH HMM/ANNS AND COUNT-BASED n-GRAMS



HMMs have been widely applied to offline handwriting recognition [El-Yacoubi *et al.* 1999a; Plamondon and Srihari 2000; Vinciarelli 2002; Bunke 2003; Koerich *et al.* 2003; Marti and Bunke 2001; Toselli *et al.* 2004; Vinciarelli *et al.* 2004] after their success in ASR. The fact that handwriting can be interpreted as a left-to-right sequence analogous to the temporal sequence of acoustic signals in speech does not seem a novelty at present but, historically, the application of HMMs came first in ASR.

Indeed, many HTR and ASR systems share now the same toolkits and basically differ in the preprocessing and feature extraction stages. Let us remark that this similarity is not limited to ASR and HTR and that other recognition tasks (some of them described in Chapter 2) may also benefit from the same systems.

The main aim of this section is twofold: on the one side, we would like to compare the preprocessing approach proposed in Chapter 14 with other alternatives [Pastor Gadea 2007] in order to obtain an empirical comparison in an unconstrained handwritten text line recognition tasks. On the other side, we also want to experiment with hybrid HMM/ANN models. These models, as described in Section 7.5.3, compute the emission probabilities for the HMMs with a neural network instead of the commonly used Gaussian mixtures. The results described here corresponds basically to [España-Boquera *et al.* 2011] which, to the best of our knowledge, was the first successful attempt to use HMM/ANNs in unconstrained offline HTR.

These experiments have been performed using count-based (a.k.a. statistical) word n-grams for language modeling, which seems a sensible choice both for comparison purposes (results reported in the literature used them too) and because they perform reasonably well.

15.1.1 Experimental setting (corpora, dictionary and LM)

The IAM database

The presented experiments have been conducted with the "large writer-independent text line recognition task" IAM database [Marti and Bunke 2002] (from now on, abbreviated IAMdb), which is described in Section A.2.1. For this first experimentation, that can be considered a baseline, we would like to thank the generosity of Moisés Pastor for sharing with us the preprocessing used in his work [Pastor Gadea 2007] and even some scripts in order to achieve a faster experimentation and to better compare the results.

The version 3.0 of this database includes over 1 500 scanned forms of handwritten text from more than 650 different writers. An example is shown in Figure 215. These forms correspond to a total of more than 13K fully transcribed handwritten lines. Acquisition was performed without restrictions on the writing style or the writing in-

strument used. The sentences have been extracted from the Lancaster-Oslo/Bergen (LOB) text corpus [Johansson *et al.* 1986] described in Appendix A.1.1.

The subset of IAMdb used in this work consists of 6 161 training lines (from 283 writers), 920 validation lines (56 writers) and 2 781 test lines (161 writers). All these data sets are disjoint, and no writer has contributed to more than one set. These partitions are the same as those used in several works by Bunke et al [Graves *et al.* 2009; Bertolami and Bunke 2008a;b] and differs slightly from the task defined at the IAM web page.[2]

Let us remark that experiments reported for IAMdb can be performed in, at least, the following four modalities and that it would be a mistake to try to compare results belonging to different ones:

ISOLATED WORDS are the result of breaking text lines into words.[3] The possibility of words broken into lines (see Figure 16) has not been taken into account. This modality has only been used, in this work, to perform experiments to combine HMM/ANNs with holistic ANN classifiers (Section 15.5) and with a bimodal corpus (Section 15.6);

LINES are the images extracted by breaking the IAM pages into lines. A line may contain several fragments of sentences (for instance, the final of a sentence and the beginning of the next one). All the experiments described in this chapter, unless otherwise stated, are performed at the line level;

SENTENCES are associated to the linguistic concept of sentence and are obtained by concatenating consecutive line fragments. Observe that this is only possible because the creators of the IAMdb supervised the text lines in order to break them into sentence fragments and labeled them in such a way to allow the extraction of sentences. A sentence is usually converted into a large artificial text line image by concatenating the line fragment constituents (it is convenient to include an space separator between them). Some preliminary experiments from our research group were performed at the sentence level [Gorbe-Moya *et al.* 2008];

PARAGRAPHS or, rather, pages, refers to the fact of recognizing an entire IAMdb form broken into lines. Jointly recognizing the set of lines benefit from a better language modeling since lines are not arbitrary and the final of one line is the perfect linguistic context to pursue the beginning of the next one. Some recent works reported in the literature are based on paragraphs and we plan to adopt them as future work.

The choice between isolated words, lines, sentences or paragraphs influence the rest of the system and, particularly, the LM.

2 Training is the same in both partitions, but our test set includes more lines.

3 Observe that this process has been performed in a semi-supervised way by IAMdb developers since geometric features can only detect ink blocks to be used as some kind of over-segmentation information (as discussed in Section 14.7) whereas the alignment with a transcription may produce more reliable results.

Table 24: Graphemes for the IAM corpus.

Lower case letters	a b c d e f g h i j k l m n o p q r s t u v w x y z
Upper case letters	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Digits	0 1 2 3 4 5 6 7 8 9
Punctuation marks	<space> - , ; : ! ? / . ' ( ) * & # +

Dictionary

A total of 87 967 instances of 11 320 distinct words occur in the union of the training, validation, and test sets.

However, excepting for some closed recognition experiments, and in order to achieve unconstrained recognition, we have used an open dictionary composed of the most frequently occurring (case insensitive) words in the material used to train the language models, described below. An open dictionary of 20K words has been used in the baseline and in the main HMM/ANN experiments, other sizes will be tried afterwards to evaluate the influence of this parameter in Section 15.1.4. Lexicons of more recent experiments (e.g. Section 15.2) are obtained in a case sensitive way.

Lexicon is modeled with 79 characters shown in Table 24: 26 lowercase letters, 26 uppercase letters, 10 digits, 15 punctuation marks, the space, and a character for garbage symbols.

Language model

A word bigram language model was trained with three different text corpora: the LOB corpus [Johansson *et al.* 1986] (excluding those sentences that contain lines from the test set of the IAM database), the Brown corpus [Francis and Kucera 1979], and the Wellington corpus [Bauer 1993] (see also Appendix A.1.1).

In order to cope with the fact that lines are fragments of sentences, each sentence from the corpus has been randomly broken into fragments to resemble lines.[4] All this text is supplemented with the training lines from IAMdb. Then, the final training material comprises:

- Sentences: 51 560 LOB sentences (2 134 sentences which contained IAM test lines were eliminated), 51 763 Brown sentences, and 20 592 Wellington sentences;
- fragments of sentences to resemble lines: more than 400K lines randomly obtained from the above set of sentences;
- Lines: finally, the 6 161 IAM training lines were also added.

The bigram language model used in the recognition systems was generated using the SRI Language Modeling Toolkit [Stolcke 2002] with the modified Kneser-Ney back-off discounting. These models, obtained in the `arpa` format and later transformed into our `lira` representation (described in Section 9.3.1) to be used with our April toolkit.

4 Other approaches to cope with lines are applied in subsequent experiments and described below.

15.1.2 Baseline experiments with HMM/GMMs

Although our main interest are hybrid HMM/ANN models, we consider convenient to make some experiments with HMM/GMM models to obtain a baseline.[5] The experiments have been conducted using continuous density HMMs with diagonal covariance matrices[6] of 64 Gaussians in each state, and with a left-to-right topology without skips (as described in Section 7.3.1). The 79 models were trained with the HTK toolkit [Young *et al.* 1993]. They have been tested with our April toolkit using the word external oversegmenter described in Section 14.7 and the two stage decoder (that comprises a DAG generator described in Section 11.1, and the DAG decoder described in Section 12.2). This decoder has been used in most other experiments as well replacing the computation of HMM emission where appropriate.

Relating the number of states of HMMs, [Zimmermann and Bunke 2002b] investigates the use of three different schemes to optimize this value on linear left-to-right HMMs, as described in Section 7.3.1. The experiments conducted here correspond to the first and more simpler approach considering the HMMs to have the same length for every grapheme, although, as a future work, we plan to experiment with different number of states per grapheme. The validation set of the IAMdb was used to optimize the number of states of HMMs using the open dictionary of 20K words and a bigram language model.[7] Results are reported using the WER (as defined in Section 3.5.1). Table 25 summarizes these experiments and comparing our preprocessing with that from [Pastor Gadea 2007], whereas Figure 203 plots these results (only for our preprocessing) with a 95% confidence level interval. It can be observed that several configurations achieved equivalent statistical performance, the 8-state HMMs being the best topology.

Table 25: Tuning the number of HMM/GMM states: WER on validation set comparing two different preprocessings (Without and With our preprocessing, where Without comes from [Pastor Gadea 2007]) Table taken from [España-Boquera *et al.* 2011; Table 2].

Model	WER on Validation (%)	
	Without	With
6-state HMM/GMMs	49.7 $\pm$ 1.5	35.1 $\pm$1.6
7-state HMM/GMMs	47.2 $\pm$ 1.5	34.5 $\pm$1.6
8-state HMM/GMMs	**46.9 $\pm$ 1.5**	**32.9 $\pm$1.5**
9-state HMM/GMMs	47.7 $\pm$ 1.5	33.3 $\pm$1.5
10-state HMM/GMMs	50.3 $\pm$ 1.5	35.0 $\pm$1.5
11-state HMM/GMMs	56.3 $\pm$ 1.5	37.3 $\pm$1.5
12-state HMM/GMMs	61.9 $\pm$ 1.5	40.0 $\pm$1.6

5 These models have been previously used in HTR. Indeed, HMM/GMM had virtually eclipsed most other modeling alternatives in ASR until, perhaps, recently with the accession of deep learning techniques.

6 One of the most expected critiques, discussed below, is related to the lack of decorrelation (e.g. PCA) on features extracted with [Toselli *et al.* 2004], which is usually required by this type of GMMs.

7 LOB sentences which contained IAM validation lines were also excluded to estimate the bigram language model for the tuning experiments.

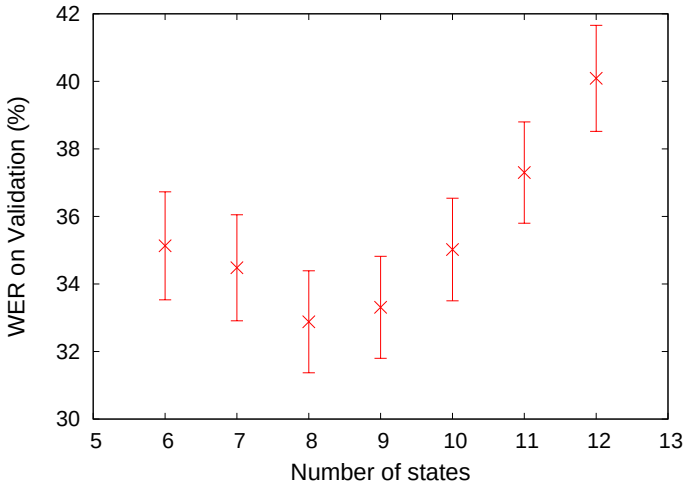


Figure 203: WER of the HMM/GMMs on the validation set varying the number of states. WER is given with a 95% confidence interval (extracted from [España-Boquera *et al.* 2011; Fig. 7]).

Table 26: Tuning the GSF: WER of the best HMM/GMMs on the validation set for different Grammar Scale Factors (Without and With our preprocessing, where Without comes from [Pastor Gadea 2007]). Table taken from [España-Boquera *et al.* 2011; Table 3].

Model	GSF	WER on Validation (%)	
		Without	With
8-state HMM/GMMs	30	47.7 ±1.5	35.6 ±1.7
8-state HMM/GMMs	35	47.2 ±1.5	33.9 ±1.6
8-state HMM/GMMs	40	**46.9 ±1.5**	32.9 ±1.5
8-state HMM/GMMs	45	47.2 ±1.5	32.9 ±1.5
8-state HMM/GMMs	50	48.0 ±1.5	**32.8 ±1.4**
8-state HMM/GMMs	55	49.0 ±1.5	33.3 ±1.4
8-state HMM/GMMs	60	50.1 ±1.5	33.8 ±1.4

Grammar Scale Factor (GSF) used for these experiments was fixed to 40. It was optimized by systematically[8] afterwards by testing values from 20 to 60 on the 8-state HMMs. Table 26 shows the performance of this experiment on the validation set using bigrams. It can be observed that performance is almost identical between a grammar scale factor of 40 and 50, with the lowest WER of 32.8% being achieved with a grammar scale factor of 50. Results for our preprocessing are also plotted in Figure 204 with a 95% confidence level interval.

8 It would be better to use the technique described in [Och 2003] (as it has been done in posterior experiments) to optimize GFS and WIP, avoiding the systematical trials.

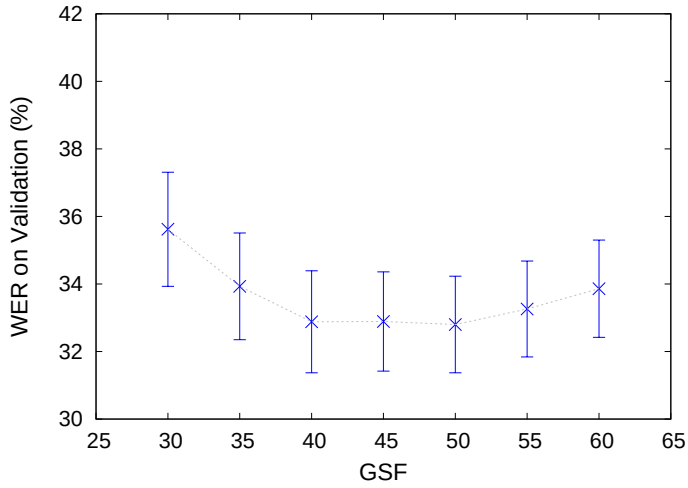


Figure 204: Tuning HMM/GMMs: WER of the HMM/GMMs on the validation set for different Grammar Scale Factors. WER is given with a 95% confidence interval (extracted from [España-Boquera *et al.* 2011; Fig. 7]).

15.1.3 Experiments with HMM/ANNs

In these experiments, graphemes have also been modeled with left-to-right Markov chains, but here a single multilayer perceptron (MLP) with one output unit for each state of the Markov chains was used to estimate the distribution probabilities. The estimates of the posterior probabilities computed by the MLP, $p(q|x)$, are divided by the prior state probabilities, $p(q)$, resulting in scaled likelihoods which are used as emission probabilities in the HMMs, as depicted in Figure 205 and explained in Section 7.5.3.

The training of the MLP is discriminant at the state level of Markov chains, since each output is optimized during training by samples of its own class as well as by samples of the other classes. Training of the whole HMM/ANN system is done using an expectation-maximization (EM) algorithm based on Viterbi alignment, using our toolkit, as follows:

1. assign an initial labeling of desired MLP outputs to every feature vector from training and validation datasets. This labeling was first generated by running a "pre-trained" HMM/ANN;
2. assign an initial non-zero value to transition probabilities of the Markov chains;
3. train the supervised ANN with the training pairs, using stochastic backpropagation with momentum and the mean square error (MSE) function. In order to monitor the generalization performance during learning and to stop training when there was no longer an improvement, the MSE on the validation dataset was

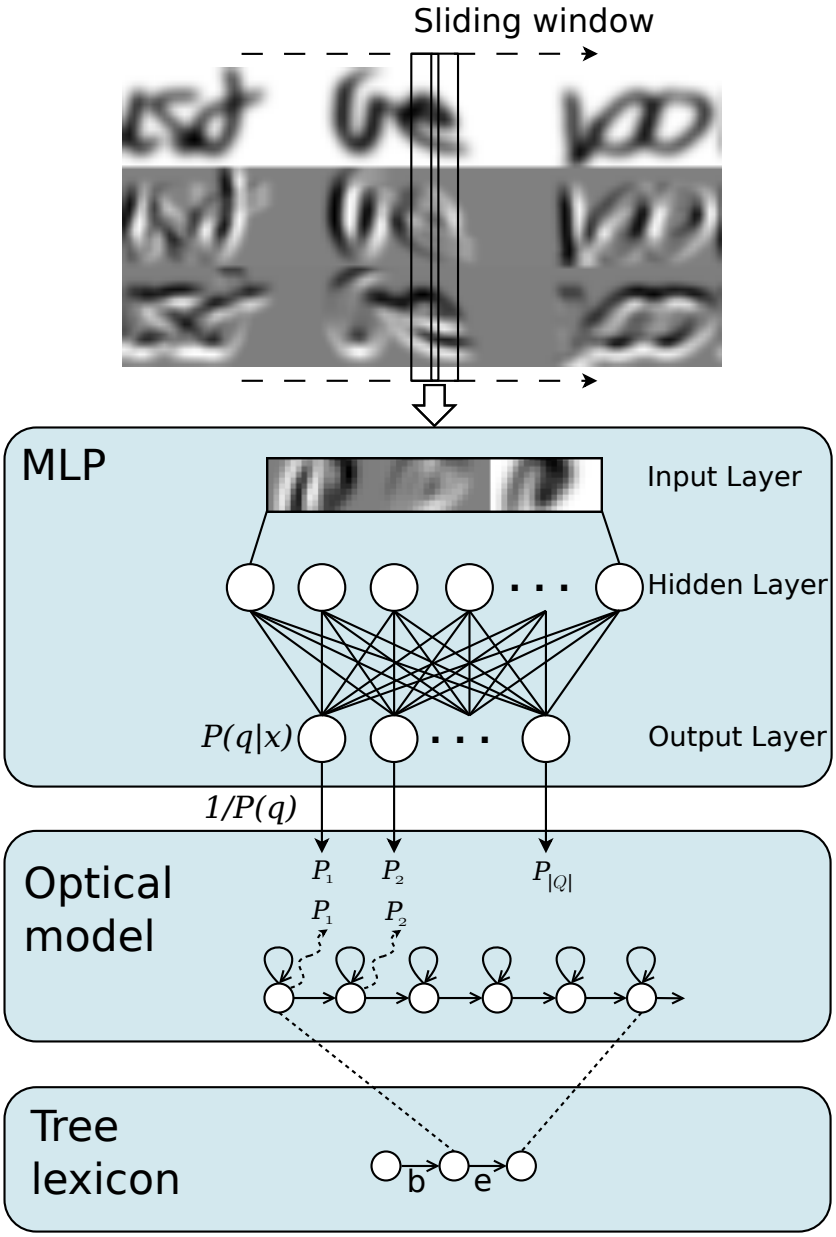


Figure 205: A scheme of the proposed HMM/ANN recognition system. The current frame plus a left and right context, is processed by an MLP. MLP outputs are divided by the prior state probabilities in order to obtain HMM emission probabilities (figure borrowed from [España-Boquera *et al.* 2011; Fig. 5]).

used as the stopping criterion. More than five million training patterns (corresponding to the 6 161 training lines) and close to 800K validation patterns (from 920 lines) composed the training and validation datasets, respectively. Due to the large time requirements to train the MLPs, we used a resampling algorithm: 300K training patterns and 200K validation patterns (randomly selected in each run) in each training epoch;

4. use the partially trained hybrid ANN/HMM models to perform a forced Viterbi alignment of the training data. This Viterbi procedure uses the class priors estimated from the relative frequencies of each class in the training data. This Viterbi alignment is used for both obtaining a new segmentation or labeling of the training and validation sets and also for counting the number of times each HMM transition has been used. These counts are used to reestimate the transition probabilities;

5. go to step 3 until convergence, that is, until the difference between two consecutive iterations is below a threshold.

Hybrid HMM/ANN models with a different number of states and different topologies and parameters of MLP were tested. In all cases, fully connections between consecutive layers were considered. Also, the MLP input always consisted of 9 consecutive feature vectors (the central feature vector and a context of four vectors at each side) giving 540 input units (the 60-dimensional nine feature vectors) normalized to zero mean and unit variance.[9] Softmax outputs units are used at the output layer. The number of output units is determined by the total number of states of the 79 optical models (from 79×6 output units for 6-state Markov chains to 79×9 output units for 9-state Markov chains), since each output unit of the MLP is related to one state of the HMMs. Hidden units used sigmoid activation. The number and size of hidden units was determined empirically by measuring the MSE on the validation set. Other parameters such as the learning rate and the momentum term were also empirically tuned with the validation data.

The WER on the validation set, using a closed dictionary from the IAM task composed of 10 353 words and a bigram language model estimated with the LOB corpus, was used as the stopping criterion. Around ten iterations of the EM training algorithm were enough for all the configurations. Figure 206 illustrates a typical evolution of the EM training, showing the evolution of the MSE on the training and validation set. The evolution of the WER on validation is also shown.

Table 27 shows the performance achieved for the different configurations of the hybrid HMM/ANN systems on the validation dataset using the same bigram LM, the same open dictionary and the same two pass decoder with word oversegmentation information that were also used in previous experiments for HMM/GMMs. Results for our preprocessing are also plotted in Figure 207 with a 95% confidence level interval. For the sake of comparison, we have conducted the same HMM/ANNs training using two different preprocessings, as was also done for HMM/GMMs. Again, our preprocessing outperforms the standard preprocessing of [Pastor Gadea 2007].

9 Estimated from the training part.

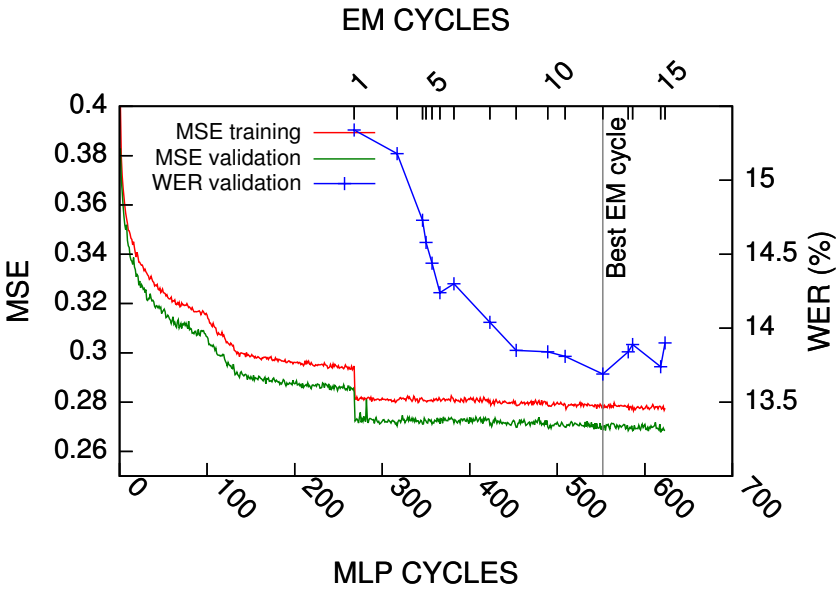


Figure 206: Evolution of the EM training algorithm for the 7-state HMMs and an MLP of two hidden layers of 192 and 128 units each. The MSE is shown for the training and validation data, along with the WER on validation of each iteration of the EM algorithm. Taken from [España-Boquera *et al.* 2011; Fig. 8].

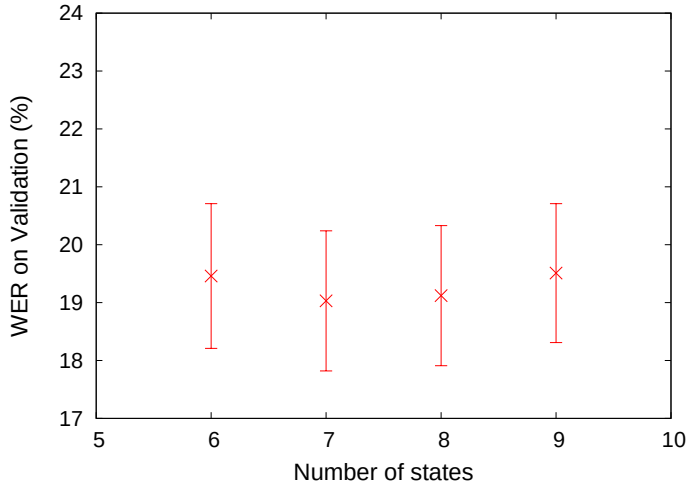


Figure 207: WER of the HMM/ANN models on the validation set varying the topology. WER is given with a 95% confidence interval (extracted from [España-Boquera *et al.* 2011; Fig. 9]).

Table 27: Tuning the topology of the HMM/ANNs: WER of the HMM/ANNs on validation (Without and With our preprocessing, where Without comes from [Pastor Gadea 2007]). Taken from [España-Boquera *et al.* 2011; Table 4].

Model	WER on Validation (%)	
	Without	With
6-state HMMs, MLP 192-128	27.8 ±1.5	19.5 ±1.3
7-state HMMs, MLP 192-128	**24.9 ±1.5**	**19.0 ±1.2**
8-state HMMs, MLP 384-128	25.0 ±1.5	19.1 ±1.2
9-state HMMs, MLP 384-128	27.3 ±1.5	19.5 ±1.2

Table 28: Tuning the GSF: WER of the best HMM/ANNs on validation for different GSFs (Without and With our preprocessing, where Without comes from [Pastor Gadea 2007]). Taken from [España-Boquera *et al.* 2011; Table 5].

Model	GSF	WER on Validation (%)	
		Without	With
7-state HMMs, MLP 192-128	6	27.0 ±1.5	21.3 ±1.4
7-state HMMs, MLP 192-128	8	24.8 ±1.5	19.6 ±1.3
7-state HMMs, MLP 192-128	10	24.9 ±1.5	19.0 ±1.2
7-state HMMs, MLP 192-128	12	**24.4 ±1.5**	**19.0 ±1.2**
7-state HMMs, MLP 192-128	14	25.0 ±1.5	19.3 ±1.2

The lowest WER was obtained by using 7-state Markov chains and an MLP topology of two hidden layers of 192 and 128 units. Further experiments were conducted with the optimized configuration: Table 28 shows the performance of this hybrid HMM/ANN system on the validation dataset using the bigram language model with different grammar scale factors. GSF has been optimized by systematically testing values from 6 to 16 on the validation text lines. Small changes in WER are observed, and the best performance, 19.0%, was achieved with a grammar scale factor of 10 or 12. Results for our preprocessing are also plotted in Figure 208 with a 95% confidence level interval.

15.1.4 Analysis of results

The performance of the optimized HMM/GMM and HMM/ANN systems on the test set is described in this section together with a comparison with the best published results. Experiments to study the influence of the dictionary on the recognizer are also carried out.

Comparing HMM/GMMs and HMM/ANNs

Table 29 shows the error rate of the recognized test lines of the IAM task using the best HMM/GMM and HMM/ANN systems. We tested each system with the open dictionary of 20K words and a bigram LM. For each recognition experiment, two performance figures were obtained: the WER and the character error rate (CER). Obviously, no parameters were optimized on the test set.

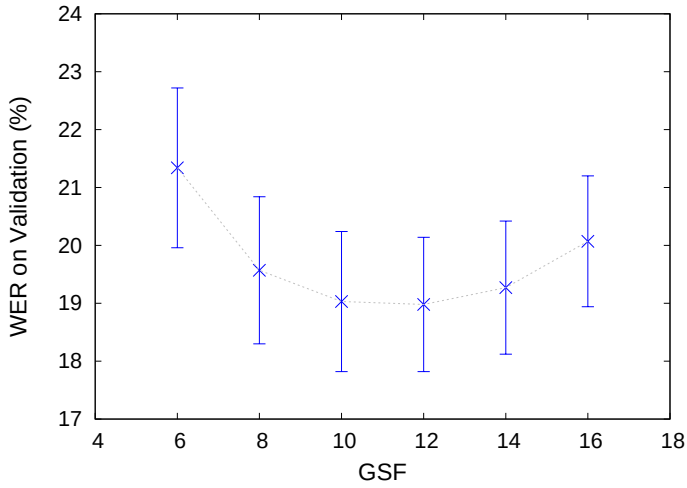


Figure 208: WER of the HMM/ANN models on the validation set for different GSFs. WER is given with a 95% confidence interval (extracted from [España-Boquera *et al.* 2011; Fig. 9]).

Table 29: Testing the systems: Error Rate of the HMM/GMMs and the HMM/ANNs (using 192-128 MLP topology) on the test set (Without and With our preprocessing, where Without comes from [Pastor Gadea 2007]). Taken from [España-Boquera *et al.* 2011; Table 6].

Best model	Results of Test (%)			
	Without		With	
	WER	CER	WER	CER
8-state HMM	54.3 ±1.0	31.7 ±0.6	38.8 ±1.0	18.6 ±0.6
7-state HMM/ANN	29.8 ±1.0	15.3 ±0.6	22.4 ±0.8	9.8 ±0.4

Our baseline experiment achieved comparative performances with state-of-the-art HMM systems: a WER of 38.8% ±1.0 with a 95% confidence interval and a CER of 18.6% in the interval (18.0, 19.2). The best result obtained with HMM/GMM, for the same task, and when these experiments were carried out, was a WER of 35.5% [Graves *et al.* 2009]. Our final hybrid HMM/ANN system achieved excellent results: a WER of 22.4% in the interval (21.6, 23.2) and a CER of 9.8% in the interval (9.4, 10.2). The hybrid system outperforms our baseline in 16 points in WER, which represents a relative error rate reduction of 42%. Similarly, the character error rate improved nearly 9 points, which represents a relative percentage of improvement that is greater than 47%. More recent comparative results, including other systems from the literature, are reported in Section 15.2.

Besides the word and character error rates, another measure to consider when evaluating a recognition engine is the decoding time, since

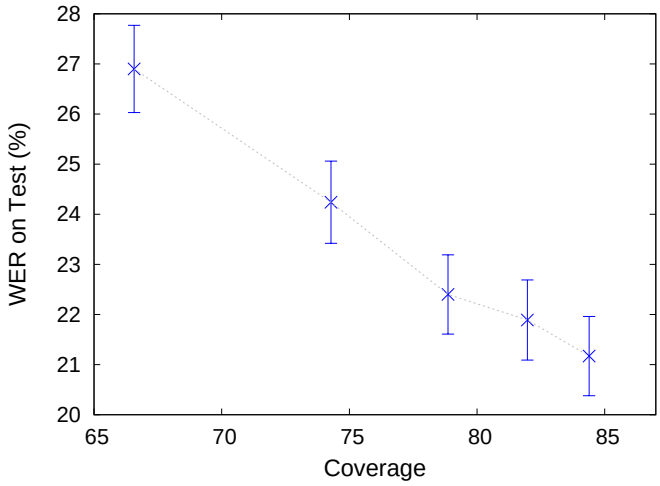


Figure 209: HMM/ANN performance with open dictionaries plotted against test set coverage. WER on test is given with a 95% confidence interval (taken from [España-Boquera *et al.* 2011; Fig. 10]).

a high value might diminish the usability of the recognition system in practical applications. Unfortunately, this time is not reported by most authors. Our hybrid HMM/ANN prototype required an average time of 0.76 seconds per word for preprocessing and 2.4 seconds per word on decoding España-Boquera *et al.* [2007]. This CPU time was measured in a single core of an Intel® Core™2 Quad CPU Q6600 @ 2.40GHz using DDR2-800Mhz memory. This time could be greatly reduced in the production stage of the recognition engine, some steps into this direction have been taken into account.

Influence of the dictionary

A series of experiments to study the influence of the dictionary size were carried out. Open dictionaries with between 10K and 30K words were generated by taking the N most frequently occurring words in the training material for the LM.

Table 30 shows the WER and CER of the test set, as well as the test set coverage when the size of the dictionary was increased. A bigram LM has been estimated for each lexicon. To give an idea of the meaning of coverage in the IAM line task, consider, for example, that with the open dictionary of 20K words, a coverage of 78.86% was achieved, that is, 21.14% of the words in the test set were out-of-vocabulary words. However, if we measure the out-of-vocabulary running-words, this figure falls to 4.99%. (The out-of-vocabulary running-words were 1 268, that is, 1 268 running-words from the 25 424 running-words in the test set were not in the lexicon.) As expected, performance increased with lexicon size and test coverage. Figure 209 plots this experiment against test set coverage.

Table 30: Influence of the dictionary size (with open dictionaries): WER of HMM/ANNs on the test set (taken from [España-Boquera *et al.* 2011; Table 7]).

Dictionary size	Coverage (%)	Results of Test (%) WER	CER
10,000	66.57	26.9 ±0.9	11.7 ±0.5
15,000	74.28	24.2 ±0.8	10.5 ±0.4
20,000	78.86	22.4 ±0.8	9.8 ±0.4
25,000	81.97	21.9 ±0.8	9.4 ±0.4
30,000	84.39	21.2 ±0.8	9.1 ±0.4

Dictionary size	Results of Test (%) WER	CER
4,953	15.5 ±0.7	6.9 ±0.4
20,000	16.8 ±0.7	7.5 ±0.4

Table 31: Influence of using closed dictionaries: WER of HMM/ANNs on the test set (table borrowed from [España-Boquera *et al.* 2011; Table 8]).

Another experiment was also carried out to study the influence of using closed dictionaries. Two closed dictionaries were generated: one containing only the 4 953 words in the IAM test line set, and another one padding the first one up to 20K words (the most frequently occurring words in the training material for the language model). The influence of using the closed dictionaries is shown in Table 31 using a 95% confidence interval for WER. In this case, a bigram language model estimated for each lexicon was also used. Not surprisingly, the closed dictionary containing only the test set words achieved the best performance. The score with the 20K word closed dictionary was still better than those reached with open dictionaries.

Comparison with other systems

Comparisons to other recognition systems, when restricted to publications using the IAM database up to the date when these experiments were performed, seems mainly restricted to [Graves *et al.* 2009], mentioned before, which presents a novel HTR system based on BLSTMs. They achieved the best published recognition rates to the date of their publication: a WER of 25.9% with bigrams. In order to compare both systems, we contacted the authors to exactly reproduce the same experimental conditions. They provided us with their dictionary and language model and we retrained our optical models using the same grapheme set and normalizing each input feature to zero mean and unit variance. Also, case differences were taken into account during recognition and in the WER computation. Surprisingly, we obtained the very same 25.9% of WER, despite the fact that recurrent neural networks and HMM/ANN are very different approaches. More recent comparisons with other systems from the literature are performed with our newer results described in the following sections.

15.2 HTR WITH HMM/ANN MODELS AND CONNECTIONIST LMS

This section describes more recent experiments that have been performed in collaboration with other members of our research group and with researchers from IAM at Bern University. The results reported here have been published in [Zamora-Martínez *et al.* 2014]. The main aim for this collaboration was to improve the results reported in previous section and to investigate whether or not a further improvement could be obtained by combining our different recognizers. The improvement obtained by our system have been achieved by taking action on two different fronts:

- improving HMM/ANN models;
- using NNLMs instead of count-based n-grams.

We refer to the reader to [Zamora-Martínez *et al.* 2014] for the description of the whole systems. Let us summarize the main similarities and differences w.r.t. the experiments from previous section:

- the same IAMdb corpus has been used, including the partitions into training, validation and test. However, the previous *trick* to deal with lines (namely, breaking fragments of sentences to resemble lines) is not applied now. It seems wiser just to train LMs as usual but ignoring the context cues (explained in Section 6.1.1) during decoding, as described in [Zamora-Martínez 2012; Section 8.6];
- the corpora for training the LMs and for constructing the lexicon is also the same, although now the dictionaries have been constructed in a case sensitive way and increasing their size. Indeed, two different dictionaries have been used now in the experiments:
  - a 103K dictionary, which corresponds to the full training lexicon;
  - a 55K dictionary, obtained after removing the singletons (that is, words that appear only once in the entire text corpus) from the one with 103K.

  This means that, even with count-based bigrams, the reported results are not comparable with those from previous section;
- although we have continued using count-based n-grams, they have not been limited to bigrams (the order has been increased up to 4-grams). Besides, although they have also been generated using the SRI Language Modeling Toolkit [Stolcke 2002], the Witten-Bell discount (WB) has been applied instead of the modified Kneser-Ney since WB has led to better PPL in validation (see [Zamora-Martínez *et al.* 2014; Table 5] for more details);
- the preprocessing stages remain basically unchanged, although some slight modifications have been performed relating the positioning in the final size normalization stage (Section 14.5);

- hybrid HMM/ANN models maintain their topology (7-state left-to-right HMMs), the same MLP topology (same hidden layer sizes 192-128) and input size (a context of 4 vectors on the left and 4 on the right, feature extraction [Toselli *et al.* 2004] set to 60 parameters per frame). They have also been trained with a similar EM algorithm using stochastic backpropagation with momentum with replacement (as with previous experiments, 300K patterns for training and 200K for validation) but, now, a weight decay regularization is also used (weight decay = 10^{-10}) and a slight perturbation Gaussian noise ($\mu = 0, \sigma = 0.015$) has been added to the input patterns during training, also for regularization purposes;

- the training of NN LMs is basically as described in Sections 6.6.1 and 9.5, but see [Zamora-Martínez *et al.* 2014; Section 5] for more details. NN LMs use the shortlist approach (using the 10K more frequent words), using an ensemble of 4 MLPs,[10] and combining them with the count-based WB n-gram models. Results on PPL (for validation data) are summarized in [Zamora-Martínez *et al.* 2014; Table 6], they systematically improve the same measures of performance of their count-based counterparts. As with count-based n-grams, context cues are not taken into account during decoding, either, due to the fact of recognizing lines;

- the same lexicon, corpus partition and LMs have been used by researchers from IAM using BLSTMs with CTC decoding instead of HMM/ANNs. They have also, in turn, improved their previous results described in [Graves *et al.* 2009]. A brief description of the techniques used in this part can be found in Section 4.4.3, although more details about the actual settings used in the current experimentation are given in [Zamora-Martínez *et al.* 2014; Section 4.1]. Let us remark that a previous comparison with their former BLSTMs ([Graves *et al.* 2009]) and with the results described in previous section ([España-Boquera *et al.* 2011]) led, surprisingly, to similar results on test.

Tables 32 and 33 show the WER and CER of IAMdb validation set for different LM configurations of BLSTM and HMM/ANN systems and by performing different combinations of count-based and feature-based (NN LMs) n-grams. In all cases, NN LMs performs better, even when these models are not combined with WB smoothed n-grams. The benefits of combining WB smoothed n-grams in the NN LM systems is not totally clear when measured on validation set, being less clear for BLSTM system. Two possible explanations are the fact that this set is substantially smaller than the test set and the fact that it has already been used to optimize the systems. The increase of the dictionary size leads to a better performance for both recognizers, although this effect seems more pronounced for the HMM/ANN system, whose performance reaches the best figures when using NN LMs with the 103K dictionary.

10 Each neural network has a different projection layer size (160, 192, 224, 256), but the same hidden layer size (200 neurons).

Table 32: Validation % WER and % CER results for the BLSTM system. Table taken from [Zamora-Martínez *et al.* 2014; Table 7].

NN LM	$\|\Omega\|$	Witten-Bell None		bigram		3-gram		4-gram	
		WER	CER	WER	CER	WER	CER	WER	CER
None	55K	–	–	18.2	7.8	18.1	7.9	18.0	7.9
bigram	55K	17.6	7.7	17.6	7.7	17.6	7.7	17.6	7.7
3-gram	55K	17.5	7.7	17.6	7.7	17.6	7.7	17.6	7.7
4-gram	55K	17.4	7.7	17.6	7.7	17.7	7.8	17.6	7.6
None	103K	–	–	17.6	7.8	17.5	7.7	17.4	7.7
bigram	103K	17.0	7.6	17.0	7.5	17.1	7.6	17.0	7.5
3-gram	103K	16.9	7.5	17.0	7.5	17.0	7.5	16.9	7.5
4-gram	103K	16.7	7.5	17.0	7.5	17.0	7.5	17.0	7.5

Table 33: Validation % WER and % CER results for the HMM/ANN system. Table taken from [Zamora-Martínez *et al.* 2014; Table 8].

NN LM	$\|\Omega\|$	Witten-Bell None		bigram		3-gram		4-gram	
		WER	CER	WER	CER	WER	CER	WER	CER
None	55K	–	–	18.4	6.7	17.9	6.5	17.8	6.5
bigram	55K	17.2	6.2	17.1	6.3	17.1	6.3	17.0	6.3
3-gram	55K	17.1	6.2	17.0	6.2	16.8	6.1	16.8	6.2
4-gram	55K	17.0	6.1	17.0	6.1	16.8	6.1	16.8	6.1
None	103K	–	–	17.4	6.3	16.9	6.1	16.8	6.1
bigram	103K	16.1	5.8	16.1	5.8	15.9	5.7	16.0	5.8
3-gram	103K	15.9	5.7	15.9	5.8	15.7	5.7	15.7	5.7
4-gram	103K	16.0	5.7	15.8	5.7	15.7	5.7	15.7	5.7

In summary, each recognition system takes advantage of NN LMs, with better recognition for the 103K dictionary. The best count-based n-grams are 4-grams LMs, for both 55K and 103K dictionaries.

Combining HMM/ANN and BLSTM/CTC based systems

Another aim of this collaborative work was to investigate if further improvements could be obtained by combining the recognizers using a generalized version of the Recognizer Output Voting Error Reduction (ROVER) system [Fiscus 1997; Stolcke *et al.* 2000].

To this end, both recognition engines were modified in order to output N-best lists instead of just the best hypothesis. These results are then combined in a weighted voting scheme using the SRILM toolkit [Stolcke 2002], selecting the 1-best hypothesis to measure the ROVER engine. Figure 210 illustrates the whole system.

The performance for the test set has been only measured for the more promising recognition configurations on validation (a trade-off between WER, CER and PPL). The results are shown with a 95% confi-

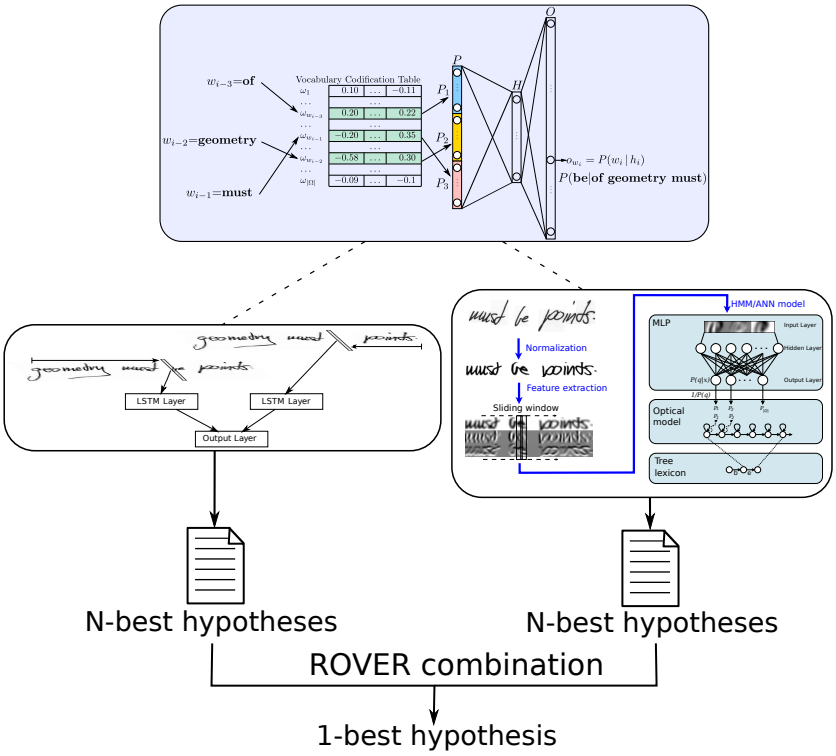


Figure 210: Scheme of the ROVER recognition system (taken from [Zamora-Martínez *et al.* 2014; Fig. 6]).

dence interval in Figure 211. In all cases, systems which use NN LMs perform consistently better than systems using only WB smoothed n-grams. The relative improvement is up to 2.7% of WER for the BLSTM system, and up to 5.2% for HMM/ANNs. Improvements are statistically significant for a confidence of 59% and 87% for BLSTM and HMM/ANN respectively. However, under a system comparison test [Bisani and Ney 2004], the improvements have more than a 99.9% probability of being statistically significant. The system comparison tests are also shown in Figure 211. Confidence intervals and system comparison tests were computed following the bootstrap technique described in [Bisani and Ney 2004].

Thus, every system improves from WB n-grams to NN LMs and the combined ROVER system outperforms the independent recognizers. A final WER 16.1% is achieved for the test set with the ROVER system, which corresponds to a further statistical significant improvement over the two best individual systems (25% relative over the best individual BLSTM system, and 20% relative over the best HMM/ANN system). Similarly, the final CER of 7.6% corresponds to a relative improvement over the individual systems of 26% and 8%, respectively. The improvement of WER from ROVER WB to ROVER NN LMs +WB is statistically significant with a 75% of confidence, but the system comparison test shows a statistical significance of more than 99.9%.

Finally, the WER and CER performance on the test set of various recognition systems reported in the literature has been collected in

Table 34: Reference systems (from [Zamora-Martínez *et al.* 2014; Table. 9]).

System	$\|\Omega\|$	WER (%)	CER (%)
Natarajan et al. (HMM)† [Natarajan *et al.* 2008]	-	40.1	-
Bertolami et al. (GHMM) [Bertolami and Bunke 2008b]	20K	35.5	-
Bertolami et al. (HMMs) [Bertolami and Bunke 2008b]	20K	32.8	-
Dreuw et al. (GHMM) [Dreuw *et al.* 2011a]	50K	29.2	10.3
TU Dortmund (HMM) [Plötz and Fink 2009]	-	28.9	-
Dreuw et al. (MLP-GHMM + M-MPE) [Dreuw *et al.* 2011b]	50K	28.8	10.1
Graves et al. (BLSTM NN/CTC) [Graves *et al.* 2009]	20K	25.9	-
España-Boquera et al. (HMM/ANN) [España-Boquera *et al.* 2011]	20K	25.9	10.5
BLSTM + NN LMs + WB	103K	21.6	10.3
HMM/ANN + NN LMs + WB	103K	20.0	8.3
ROVER + NN LMs + WB	103K	16.1	7.6

† Experiments using different subsets of the IAMdb for training, validation, and testing.

System	LM	$\|\Omega\|$	WER (%)	CER (%)	ΔWER
BLSTM	WB 4-grams	103K	22.2 ± 0.7	10.5 ± 0.4	–
BLSTM	NN LMs 4-grams	103K	21.8 ± 0.7	10.3 ± 0.4	0.4 ± 0.4
BLSTM	+ WB 4-grams	103K	21.6 ± 0.7	10.3 ± 0.4	0.6 ± 0.3
HMM/ANN	WB 4-grams	103K	21.1 ± 0.7	8.6 ± 0.4	–
HMM/ANN	NN LMs 4-grams	103K	20.5 ± 0.7	8.4 ± 0.4	0.6 ± 0.6
HMM/ANN	+ WB 4-grams	103K	20.0 ± 0.7	8.3 ± 0.4	1.1 ± 0.5
ROVER	WB 4-grams	103K	16.8 ± 0.6	8.0 ± 0.4	–
ROVER	NN LMs 4-grams	103K	16.4 ± 0.6	7.7 ± 0.4	0.4 ± 0.5
ROVER	+ WB 4-grams	103K	16.1 ± 0.6	7.6 ± 0.4	0.7 ± 0.4

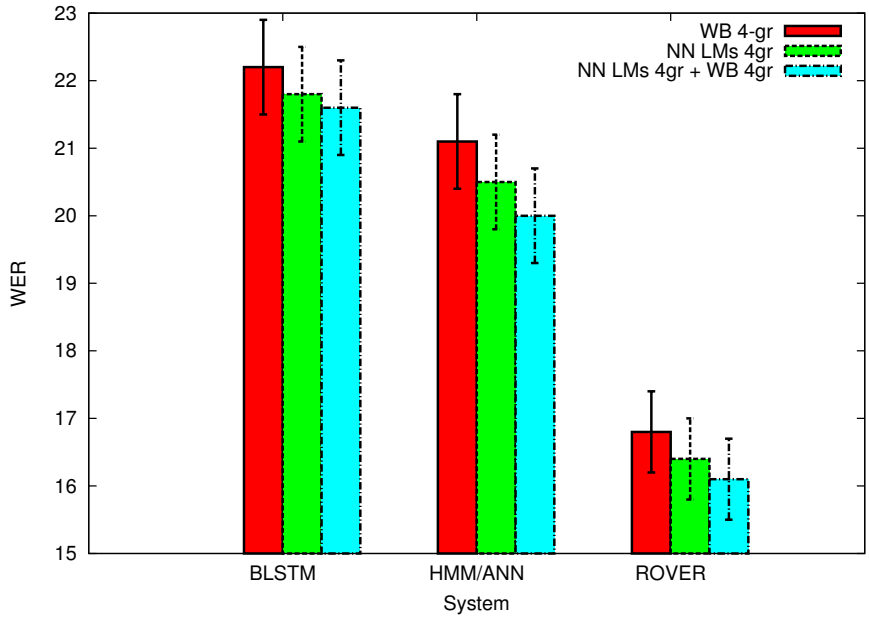


Figure 211: Test results with 95% confidence intervals. ΔWER is the WER difference with 99.9% confidence interval, comparing every NN LM system with its corresponding WB baseline (from [Zamora-Martínez *et al.* 2014; Fig. 7]).

Table 34 in order to better analyze the obtained results with respect o other systems. The table includes the baseline results obtained in the previous versions of our recognition systems, described in previous section and in [España-Boquera *et al.* 2011], as well as the previous versions of the BLSTM system presented in [Graves *et al.* 2009]. Both previous versions used the same 20K lexicon and the same bigram LM.

Careful attention must be payed when comparing systems reported in the literature, even if they use the same corpus, since they may use different LMs, dictionaries, and parameters. Nevertheless, there is a clear trend towards lower error rates. Yet, there is still room left for improvement, since the lowest possible WER using the 103k dictionary is, as far as the OOV words rate, only 2.86%.

In a final experiment the dissimilarity of both approaches has been exploited by successfully combining the N-best outputs. The obtained results were, to our knowledge, the lowest word and character error rates reported up to the date of submission of [Zamora-Martínez *et al.* 2014] on the challenging offline IAM task. The final WER of 16.1% means a relative reduction by 20% over the best individual system and a relative reduction of 38% over the best previously published reference system. The CER is reduced by 8% over the best individual system and by 25% over the best result published until this moment.

15.3 HTR BY USING CNNS FOR PREPROCESSING

using raw values

As pointed out in the previous chapter (in Section 14.6.1), connectionist methods are able to extract features from raw values or, in our case, from preprocessed image lines.

More particularly, this work proposes the use of Convolutional Neural Networks (CNNs) for this purpose. Indeed, CNNs have already been used in our research group for several related applications [Pastor-Pellicer *et al.* 2015c;a] and one of the lines of research of the PhD of Joan Pastor-Pellicer is the use of these techniques to improve in some ways the preprocessing ideas proposed in Chapter 14 (e.g. [Pastor-Pellicer *et al.* 2014; 2015a]) as well as to extend them from text line preprocessing to document layout analysis and, in particular, text line extraction.

In this context, the idea of using raw pixels from the preprocessed text line images (which is really easy due to the fact that these images have a fixed height so that the sliding window approach works *as is*) comes naturally when using CNNs in related tasks. The preliminary results obtained by retraining HMM/ANNs with ANNs that include some CNN layers (convolution and max-pooling) before the MLP have been very impressive. Although these results are not directly due to our own experimentation, but rather to a collaborative work leaded by Joan Pastor, we consider relevant to put them here since they share the same decoders, HMM/ANN training algorithms as well as most preprocessing stages described and implemented in our work. We can therefore consider that these results provide indirect evidences of the validity and the goodness of the techniques described here.

These preliminary results have been performed using the same LMs (both count-based and NN LMs), lexicon (103K) and text line image corpus used in the previous section. However, the preprocessing used in these experiments is the same as in [Pastor-Pellicer *et al.* 2015a] which slightly differs from our in the extraction of the reference lines of the main body area.[11]

It is noteworthy the fact that the number of frames when using the feature extraction from [Toselli *et al.* 2004] depends on the actual height of the preprocessed image and in the number of cells. Using a size normalization of 42 pixels height and using a feature extraction with 20 cells,[12], a frame is produced every (roughly) two pixel columns. In order to maintain this cadence, the HMM/CNN processes the text line image by advancing the sliding window two pixels each time.

A left-to-right HMM without loops with 7 states has also been maintained. Relating the window size, it has been increased as was also done in the most recent experiments with HMM/ANNS [Pastor-Pellicer *et al.* 2015a] and is now of size 42×42 pixels.[13] Although several CNN topologies have been tested, one of the configurations that have led to better results comprises a sole convolution layer *without max-pooling*. This convolution layer processes the 42×42 window using 6 kernels of size 6×6 with stride $+3 + 3$ leading to a set of $13 \times 13 \times 6 = 1014$ features that feeds a MLP with two hidden layers of sizes 512 and 128 using the Rectified Linear Unit (ReLU) activation [Nair and Hinton 2010] and a softmax output layer of size $79 \times 7 = 553$.

One of the motivations that made us try this simpler topology was the fact that, in principle, even a sole convolution layer might suffice to compute features similar to those described in [Toselli *et al.* 2004], which is the feature extraction technique used so far in our experiments. The obtained results are reported in Table 35.

Table 35: Results with HMM/CNNs on test set.

	Results of Test (%)		
LM	WER	CER	SER
4-grams	17.17	6.32	65.28
NN LMs	16.29	6.01	63.59

These results should have to be compared with the HMM/ANN counterpart of previous section (20.0 WER, reported in Table 34), which mean an important improvement. It is expected that further experiments and that the combination of this system with others, as done in previous experiments, outperforms the best figure of merit reported in Table 34. As future work, and in order to compare these results with some of the newest ones, we should also move from lines to paragraphs.

11 Instead of extracting and classifying interest points, the main body area is estimated by means of DP from a probability map generated by a CNN.

12 Which corresponds to $20 \times 3 = 60$ parameters, since vertical and horizontal derivatives are also included.

13 A left context of size 21 and a right context of size 20, this correspond, in we used the former feature extraction, to a context of size 10 frames instead of the former context of 4 size.

Table 36: Dictionaries and out-of-vocabulary (OOV) words rate in the IAM validation and test sets for each dictionary size.

		OOV rate (%)	
Dictionary	$\|\Omega\|$	Val	Test
Dictionary removing singletons	55K	4.1	3.7
Dictionary with the full vocabulary	103K	3.2	2.9

15.4 LEXICON-FREE RECOGNITION

The recognition systems described so far are all driven by a lexicon. They are unconstrained recognition systems in the sense that this lexicon is constructed without any information relating the actual words present in the test set. They have to deal with the problem of out-of-vocabulary (OOV) words even when using lexicons with tens (or even hundreds) of thousands of words. Words not belonging to the lexicon cannot appear at the recognizer output, so they are sure errors unless some technique for dealing with OOVs is taken. Some ideas to this respect are described in Sections 6.8 and 10.7 (see Figures 86 and 138).

It is also possible to deduce a lower bound on the achievable WER by measuring the OOV rate. Table 36 shows, for illustrative purposes, the OOV rate for the dictionaries used in experiments of Section 15.2.

However, this estimate per se does not take into account the fact that these errors often propagate to neighboring words by means of the LM so that later processing stages cannot usually recover from them. Moreover, OOV words are often content words – nouns, verbs, adjectives, and adverbs – [Bisani and Ney 2005]. Some authors have proposed methods to detect OOV words, and they can be taken into account in the LM, as mentioned in Section 6.8 where the use of subword based LMs is also proposed as a way to palliate this problem.

Although we seek to combine subword-based with word-based LMs and some recent works point out in this direction, a simpler possibility consists in relying only on character-based LMs. This approach [Brakensiek *et al.* 2000; Brakensiek A 2001; Wienecke *et al.* 2002; Bisani and Ney 2005; Schambach 2009] is often called lexicon-free, that is, the explicit use of a lexicon is avoided. *lexicon-free*

Character n-grams are able to learn, to some degree, the words and sequence of words appearing in the training corpus (specially for high values of n), and also to model words not belonging to that corpus.

The aim of this brief section is to describe some preliminary results in this direction performed, as with most other experiments, in collaboration with other members of our research group. Since the main focus of these experiments was on the use of sub-word based LMs and, particularly, on the use of NN LMs, they are more precisely described in [Zamora-Martínez 2012; Section 8.8] from this LM point of view, although they have been previously published in [Zamora-Martínez *et al.* 2010b]. We refer here the results of [Zamora-Martínez *et al.* 2010b] mainly as a way to provide more evidences supporting the suitability of our preprocessing and decoding techniques to this particular scenario (all these experiments have been performed with our *aims*

Table 37: NN LM topologies and SRI LM size (taken from [Zamora-Martínez *et al.* 2010b; Table 3]).

			NN LM size	SRI LM size
n-gram	P	H	Number of weights	Number of parameters
2	1·20	128	13,008	2,650
3	2·20	300	36,380	25,895
4	3·20	300	42,380	137,820
5	4·20	400	64,480	471,893
6	5·20	400	72,480	1,213,587
7	6·20	400	80,480	2,544,165
8	7·20	510	112,790	4,558,532

April toolkit whose decoders are described through this work). Unfortunately, one of the extensions[14] described in this work specifically related with lexicon-free NN LMs has not been tested in these experiments: it is related to the efficient evaluation of NN LMs specialized for small vocabularies, as described in Section 9.5.3.[15] However, since these experiments do not affect the PPL of sub-word models nor the figures of merit (WER and CER) of experiments, the simple computation of Section 9.5.3 describing the theoretical speed up achievable by this technique can be obtained in a quite straightforward way.

Relating the training of LMs, the training material is the same as in 15.1 (i.e. including the fragments of sentences obtained by randomly breaking corpus sentences into fragments) since, although this section appears after the previous description on word-based NN LMs, it was chronologically performed between [España-Boquera *et al.* 2011] and [Zamora-Martínez *et al.* 2014]. This also explains why count-based n-grams (estimated with SRI toolkit [Stolcke 2002]) are also smoothed using the modified Knesser-Ney discount.

The rationale for using NN LMs in this work is motivated not only by their better performance in terms of test set PPL but for their much better scalability when augmenting the n-gram order, as illustrated in Table 37.

Relating the training of NN LMs, it was similar to the experiments described in Section 15.2. The stochastic version of backpropagation with momentum term and weight decay algorithm is used to train the neural networks for language modeling. Learning rate was set to 2×10^{-3}, momentum term was set to 10^{-3} and a weight decay of 10^{-9} was used. The error function was the cross-entropy. Hidden layers used, in this case, the hyperbolic tangent activation function and the output layer used, as expected, the softmax activation function. Different topologies were tested, depending on the size of the input, as described in Table 37.

14 It was proposed by me, around 2009, when performed these experiments and much before than [Devlin *et al.* 2014], where a similar technique can now be found.

15 There are also other related proposals relating OOVs in this work that we could not deepen either because of lack of time, as those mentioned in Section 6.8.

Table 38: Validation WER (%) for different n using SRI LMs and NN LM plus SRI model combinations, following equation (15.1). The combination coefficient α is also shown (from [Zamora-Martínez *et al.* 2010b; Table 4]).

n-gram	SRI	NN LM	α
2	46.5	46.2	0.313
3	39.3	35.0	0.118
4	28.8	26.0	0.102
5	25.4	21.1	0.089
6	24.9	19.1	0.123
7	24.5	19.1	0.183
8	23.4	18.5	0.216

Table 39: Test WER (%) for different n using SRI LMs and NN LM plus SRI model combination (from [Zamora-Martínez *et al.* 2010b; Table 5]).

Model	Test WER (%)	Test CER (%)
SRI	30.9	13.8
NN LM	24.2	10.1

The final NN LM is linearly combined with the count-based n-gram as follows:

$$p(S) \approx \prod_{i=1}^{|S|} \left(\alpha\, p_{NN}(s_i|h'_i) + (1-\alpha) p_{SRI}(s_i|h'_i)\right) , \tag{15.1}$$

where:

- $h'_i = s_{i-n+1} \ldots s_{i-1}$ is the context of both n-grams;
- $p_{SRI}(s_i|h'_i)$ is the probability computed with the SRI n-gram;
- $p_{NN}(s_i|h'_i)$ is the probability computation of the NN LM n-gram;
- $\alpha \in [0, 1]$ is computed with the `compute-best-mix` command of the SRI toolkit. The computed values are shown in Table 38.

Relating the evaluation of NN LMs, although the technique from Section 9.5.1 based on pre-computing and storing softmax normalization constants was used in this experiment, this was probably unnecessary due to the small output size.

System evaluation

Hybrid HMM/ANNs used in these experiments are similar to those from Section 15.1.3. Table 38 shows the WER on the validation partition of the IAM lines task for different character n-grams (values of n from 2 to 8) using the SRI LMs and the NN LMs. Same values are graphically displayed in Figure 212. As it can be observed from these figures, the use of a NN LM in the recognizer system clearly outperforms the use of count-based LMs alone, for any n-gram order.

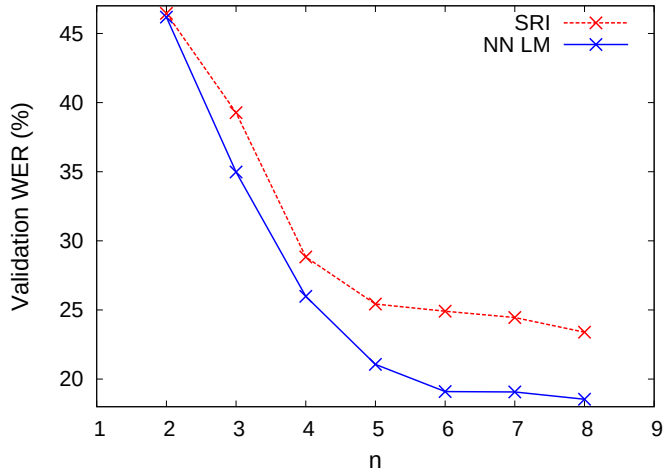


Figure 212: Validation WER (%) for different character n-gram language models (taken from [Zamora-Martínez *et al.* 2010b; Fig. 3]).

As long as the value of the n-gram increases, recognition performance improves for both LM types, very fast until 6-grams and more slowly till 8-grams. Table 39 shows the performance of the best LMs (8-grams) for the test partition of the IAM task. The CER is also shown. As it can be observed, we obtain a 24.2% WER using the NN LM and a WER of 30.9% using only the SRI LM. Similar improvement in CER is also observed when using the NN LM.

It is worth noting that the achieved result outperforms the very best results obtained for the same task (and the same test partition) at the moment of submitting this work: a WER of 25.9% reported in [Graves *et al.* 2009] obtained with BLSTMs based on the CTC lexicon-driven approach, where the lexicon was constructed with the 20K most frequent words of the LM corpora.

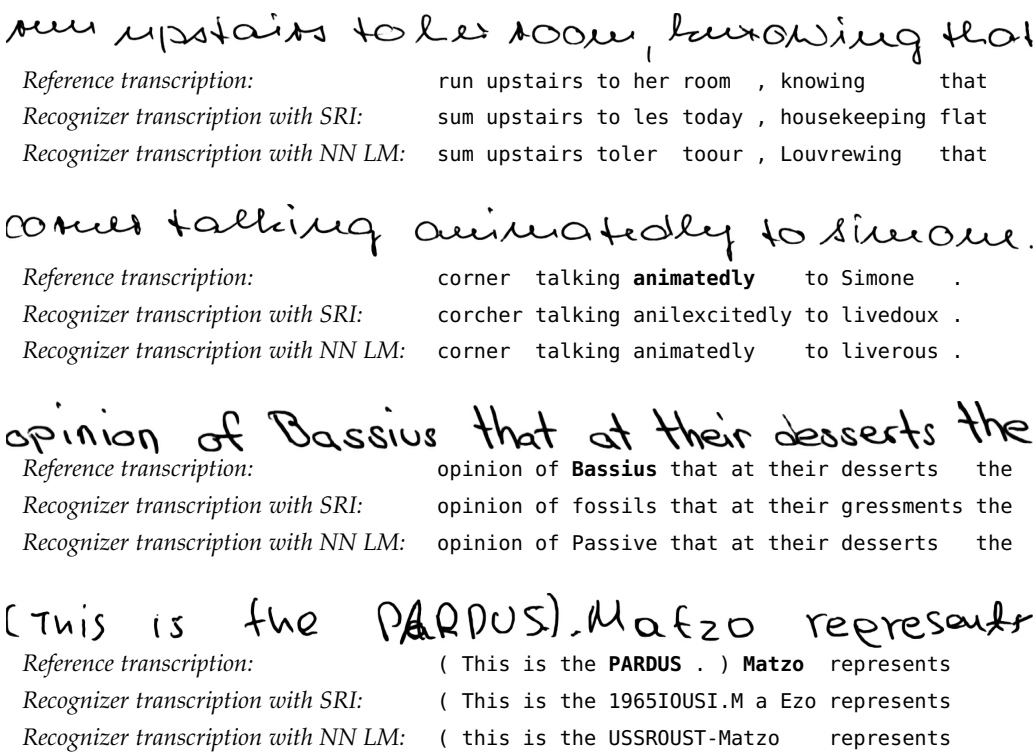
OOV word analysis

implicitly learns words

A clear advantage of character n-grams is their ability to model sub-lexical grapheme sequences leading to a lexicon-free approach. As a direct effect of this, character n-grams can very effectively recognize OOV words which, with a lexicon-driven approach, would be sure errors. It is remarkable that, for large values of n these models are able to learn implicitly whole words of the training corpus, and to some degree, even to learn word bigrams.

able to recover OOVs

Figure 213 shows some transcription examples where OOV words have been recognized. We have measured the OOV accuracy in the test partition for the final experiments (see Table 36): from a total of 544 OOV words which appear in the test partition, up to 34% were correctly recognized using the character NN LM.



Reference transcription:	`run upstairs to her room , knowing that`
Recognizer transcription with SRI:	`sum upstairs to les today , housekeeping flat`
Recognizer transcription with NN LM:	`sum upstairs toler toour , Louvrewing that`
Reference transcription:	`corner talking **animatedly** to Simone .`
Recognizer transcription with SRI:	`corcher talking anilexcitedly to livedoux .`
Recognizer transcription with NN LM:	`corner talking animatedly to liverous .`
Reference transcription:	`opinion of **Bassius** that at their desserts the`
Recognizer transcription with SRI:	`opinion of fossils that at their gressments the`
Recognizer transcription with NN LM:	`opinion of Passive that at their desserts the`
Reference transcription:	`( This is the **PARDUS** . ) **Matzo** represents`
Recognizer transcription with SRI:	`( This is the 1965IOUSI.M a Ezo represents`
Recognizer transcription with NN LM:	`( this is the USSROUST-Matzo represents`

Figure 213: Some examples of decoding outputs. OOV words are bold marked (figure from [Zamora-Martínez *et al.* 2010b; Fig. 4]).

Coverage of the character n-gram language model

Finally, we measured the coverage of the final character NN LM for 8-grams, in order to analyze how often the highest n order value is used, and how often the system needs to descend to a lower n value, when only the best recognition hypotheses are considered. Table 40 shows how often the 8-gram model is used, how often the 7-gram model is used, and so on. It can be seen that 8-grams are used in close to 90% of the cases. This leads us to think that there is still room to further increase the value of n, training bigger n-grams.

n	% coverage recognition result
2	0.0
3	0.1
4	0.6
5	1.7
6	3.3
7	5.2
8	89.1

Table 40: n coverage for NN LMs for best recognized hipotheses.

Some conclusions

We can conclude, from these preliminary experiments, that HTR based on sub-word (character) LMs can achieve recognition results competitive with state-of-the-art lexicon-based approaches while allowing, in principle, recognize test words that were not present in the vocabulary used in the training data and that constitute sure errors in lexicon driven approaches.

Among the main advantages of NN LMs for this particular task, their size is not affected by the corpus size and, also important, they grow very gracefully with the n-gram order. The first property does not hold for lexicon-based approaches whereas the second one is not achieved by count-based n-grams. On the other side, the NN LM represents the tokens in a continuous space, thus allowing a better smoothing as can be observed when comparing SRI and NN LM n-grams models using the same optical models.

15.5 COMBINING HMM/ANNS WITH HOLISTIC ANN

This section reports some experimentation done with a combination of generative and discriminative segmental models. This idea was proposed in Section 7.7 and it can be applied at several levels: in the case of HTR, it is possible to apply, for instance, at the character and at the word level. The work is preliminary because it has been limited to isolated word recognition. Our aim, as future work, is to extend this idea to the general case of continuous (e.g. line/paragraph) HTR. Indeed, we have pointed out some indications on how the implemented DAG generators of Chapter 11 can be adapted to this scenario.

isolated word recognition

The idea of performing a discriminative classification at the word level comes from the empirical observation that the recognition of short words behave considerably worse than the longer ones and that holistic methods can be used to classify them. It is curious that this observation, that we have remarked in HTR (as described below), has already been observed in ASR [Saon and Padmanabhan 2001]. Besides, since short words are quite frequent, this may have a noticeable impact. Relating prior art, [Bahl *et al.* 1989b] describes the special modeling of *function words*. Otherwise stated: the idea of using special, dedicated models, for a subset of the lexicon is not new, although the combination of holistic connectionist techniques with general HMMs, in HTR, seems novel to the best of our knowledge.

applied to a reduced subset of short words

The experiments described in this section has been done, as most other ones, in collaborative work with members from our research group and have been published in [Zamora-Martínez *et al.* 2010a], although some previous experiments were done in order to participate in the Icdar 2009 handwriting recognition competition where we obtained the second position [Grosicki and El Abed 2009]. The experiments for this competition where performed using the French RIMES database [Augustin *et al.* 2006] (see also Appendix A.2.4). They have

second position in 2009 Icdar competition

Table 41: Validation WER (%) with HMM/GMMs (left) and HMM/ANNs (right). Table taken from [Zamora-Martínez *et al.* 2010a; Table 1].

HMM	Validation (%)
6 states	37.5
7 states	35.7
8 states	33.1
9 states	**32.1**
10 states	33.3
11 states	36.0
12 states	37.7

HMM/ANN	Validation (%)
6 states, MLP 192-128	26.2
7 states, MLP 192-128	24.5
8 states, MLP 384-128	**21.9**
9 states, MLP 384-128	22.7

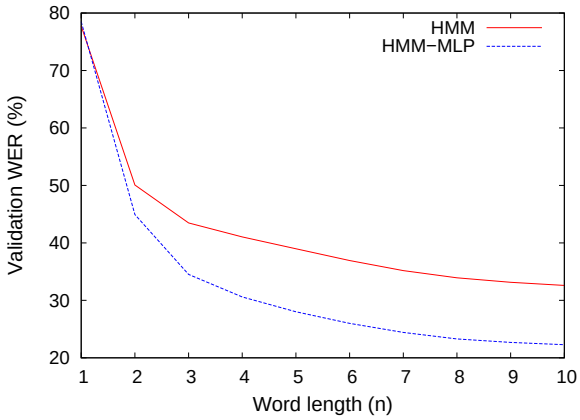


Figure 214: WER for words in the validation set with length $\leqslant n$ (taken from [Zamora-Martínez *et al.* 2010a; Fig. 2]).

been later adapted to the offline IAMdb afterwards. The description of this work is based on this last version.

Relating HMM/GMM and HMM/ANN models used in the experiments, they are essentially similar to those used in Section 15.1.3 and in [España-Boquera *et al.* 2011], although the models used here belong to an ancient, preliminary version and performs slightly worse, as can be observed in Table 41. The best WER achieved for the validation set is 21.9%, with a 8-state HMM/ANN using two hidden layers of sizes 384 and 128 for the MLP, and 32.1% with a 9-state HMM/GMM.

As mentioned before, the analysis of the results of these recognizers revealed that many errors were caused by short words. In order to confirm this observation, the validation WER for words containing between 1 and n characters has been obtained and plotted in Figure 214, showing that shorter words have an error rate far greater than the global error rate shown in Table 41.

The way to tackle this problem has been to train a classifier specialized for short words and to combine it with the initial recognizers. Let us see how this technique have improved the system performance.

MLP based holistic classifier

MLPs can be used as holistic classifiers, processing each word as a whole instead of segmenting it into smaller units [Madhvanath and Govindaraju 2001; Castro *et al.* 2005; Ruiz-Pinales *et al.* 2007]. This possibility was discussed in Section 3.1.1.

The input to the neural network may consists of features obtained from the image or even the pixel values. The output comprises a score for each possible word of the lexicon using the softmax activation function at the output layer.

reject option

Since the purpose of these models is to recognize a reduced subset of the lexicon, they must necessarily have the ability to reject word images outside that subset. For this reason, instead of trying to minimize the error rate (which would be achieved in a trivial way by rejecting all the inputs), the training process tries to maximize the harmonic mean between precision and recall, known as F-measure:

$$P = \frac{\text{correct}}{\text{total accepted}} \quad R = \frac{\text{correct}}{\text{correct+incorrectly rejected}} \quad F = \frac{2 \cdot P \cdot R}{P + R}$$

A work [Pastor-Pellicer *et al.* 2013] much more recent than these experiments proposes the use of the F-measure as the error function to train neural networks. This technique could be used in future experiments but, for the moment, the F-measure is used as stopping criterion,[16] which is the reason why the rejection of samples must be fixed a priori and used during training and recognition: an input sample is accepted if the most probable outputs have a combined probability mass greater than or equal to a given threshold. A threshold of 0.7 for the sum of the probabilities of the two most probable words has been selected after some preliminary experiments.

curse of dimensionality

The MLPs used in this work receives as input the pixels of the word image after being centered and rescaled to a fixed size (60×30 pixels). This approach is only feasible when the input layer and the lexicon are small. Otherwise, the variability of the inputs and the huge number of parameters of the model would require too many training samples.

ensemble of MLPs

Two kinds of inputs have been studied: the original and the preprocessed word images (using the same techniques of Chapter 14 as with continuous HTR). The purpose of using different representations is twofold. On the one hand, using the original images makes the system more robust against possible preprocessing errors, given that the preprocessing method was devised to deal with whole lines instead of individual words. On the other hand, using different inputs can be useful to make the classifiers less correlated so that they can also be combined.

In order to obtain a reduced lexicon for the holistic classifiers, only words with three or less characters with a minimum of 40 occurrences in the training set were considered, resulting in a lexicon of 56 words.

During recognition, the word class is unknown, so the criterion needed to decide whether an image is to be classified by the holistic MLP must rely uniquely on image features. An experimental study

16 In order to control overfitting during the MLP training, the validation set has been split into two subsets, one for stopping the MLP training, and another one for choosing the best network topology and training parameters.

Table 42: Precision (P), recall (R) and F-measure (F) of the holistic classifiers for validation (%). Table taken from [Zamora-Martínez *et al.* 2010a; Table 2].

	Validation 1			Validation 2		
Input image	P	R	F	P	R	F
Original	76.3	67.5	71.6	79.1	64.7	71.1
Preprocessed	77.3	73.0	**75.1**	76.5	77.0	**76.8**

on the training set has shown that selecting the images whose width is lower than or equal to 150 pixels accepts a reasonable percentage of short words while accepting a very low number of words outside the reduced lexicon, which should be rejected by the classifier.

Two different kinds of MLP have been trained, differing only in their inputs (original images or preprocessed images). In both cases, the input image is centered and rescaled to a fixed size of 60×30 pixels. In this way, every MLP has 1,800 neurons in the input layer. The backpropagation algorithm with momentum term has been used to train the MLPs, shuffling the training samples in each epoch. The following parameters have been tested:

- number of neurons in the hidden layers: (256, 128), (192, 128), (192, 192), (256, 192), (128, 128) and (128, 64). All the considered topologies had two hidden layers using the hyperbolic tangent as the activation function;
- learning rate and momentum term, as given in the following pairs: (0.0075, 0.004), (0.01, 0.002), (0.05, 0.01) and (0.005, 0.0001);
- connection weights and neuron biases have been initialized using random values in the interval $[-0.08, 0.08]$.

There is not an output neuron representing the unknown word, so the images of words not belonging to the reduced lexicon have been used to train the MLPs setting their target output to $1/|classes|$ for each output neuron, because the softmax activation function requires that all outputs sum to 1. This is consistent with the rejection criterion based on the probability of the two most probable classes.

For each combination of parameters, an MLP was trained until the best F-measure for the first validation subset did not improve for 200 epochs. Once all the MLPs were trained, the MLP which achieved the best F-measure for the second validation subset was selected. The results achieved for the best holistic classifier of each input type (original and preprocessed images) on the validation set are shown in Table 42.

Classifier ensembles

Given an input image, the following method has been used to combine the HMM-based recognizers with the holistic recognizers:

1. the ordered list of the N most probable words is obtained from the general recognizers (HMM/GMM or HMM/ANN);

Table 43: Validation WER combining HMM-based and holistic recognizers (taken from [Zamora-Martínez *et al.* 2010a; Table 3]).

Model	Validation WER (%)
HMM	32.1
HMM + holistic (orig)	27.4
HMM + holistic (prep)	27.9
HMM + holistic (both)	**25.5**
HMM/ANN	21.9
HMM/ANN + holistic (orig)	18.1
HMM/ANN + holistic (prep)	18.9
HMM/ANN + holistic (both)	**17.1**
HMM + HMM/ANN + holistic (both)	22.6

2. when the image width is smaller than 150 pixels, the image is also classified with the holistic MLPs. For each MLP which accepts the sample, the ordered list of the N most probable words is obtained;
3. all previous recognizer outputs are combined by means of the Borda voting method [de Borda 1781; Verma *et al.* 2001]: each word is assigned a score based on the output order (N points for the most probable word, 1 point for the least probable one), and then the words are sorted according to their total score.

In order to be able to vary the relative importance of the recognizers, an extension of the Borda count method has been considered which uses two additional parameters per recognizer:

- a constant value which is added to the scores assigned by the recognizer; and
- a weight that multiplies the previous result.

After some experiments with this approach it was observed that the best results for the validation corpus were obtained adding the recognizer scores without any weighting. These results are shown in Table 43.

Taking into account the good results obtained by the ensembles, another ensemble containing all the classifiers was tested. Unfortunately, the results were worse than the HMM/ANN by itself (22.6% WER for the validation set using the unmodified Borda count).

Finally, the test set recognition error has been computed for the best classifier ensemble on validation, as well as for the original classifiers (see Table 44). The combination of the holistic recognizers with the original ones achieves a significant improvement, between 5 and 6 absolute points, of the word error rate. The best result obtained is 22.1% WER for the test set, with the HMM/ANN and both holistic classifiers.

We can conclude that the use of a holistic classifier specialized in short words significantly improves the best results achieved by both HMM/GMMs and HMM/ANNs. However, the combination of all three recognizers does not outperform even the HMM/ANN alone, probably because HMM/GMMs and HMM/ANNs are very correlated each other.

Table 44: Test WER (%) combining the recognizers with the holistic classifiers (taken from [Zamora-Martínez *et al.* 2010a; Table 4]).

Model	Test WER (%)
HMM	38.6
HMM + holistic (both)	32.1
HMM/ANN	27.6
HMM/ANN + holistic (both)	22.1

15.6 SOME (ISOLATED WORD) BIMODAL EXPERIMENTS

This section presents a bimodal recognition engine which combines our previous offline recognition system with the online preprocessing described in Section 14.8. One of the aims of this section is to show the suitability of the online adaptation of our preprocessing.

isolated word recognition

This work is based on the biMod-IAM-PRHLT corpus which is a bimodal dataset of online and offline isolated handwritten words described in [Pastor *et al.* 2009] and also in Appendix A.2.3. Indeed, the proposed engine entered the "Bi-modal Handwritten Text Recognition" contest organized during the 2010 ICPR.

The recognition system is based on HMM/ANNs for both offline and online recognition, using the following configurations:

OFFLINE the same HMM/ANN models of [España-Boquera *et al.* 2011], described in Section 15.1.3, have been used, namely: a 7-state HMM/ANN using a MLP with layer sizes 192-128. The input was composed of the current frame plus a context of 4 frames at each side. It is worth noting that these models were trained with the training partition of IAMdb;

ONLINE models were trained with more data than that given in the bimodal corpus: the IAM-online training partition [Liwicki and Bunke 2005] (see Appendix A.2.2) was also used. Topologies of HMMs and MLP were just the same as the offline HMM/ANN models, but with a wider context at the input layer of the MLP: 12 feature frames at both sides of the actual input point.

Combination

The scores of the 100 most probable word hypothesis were generated for the offline and the online samples using the corresponding engines. The final score for each bimodal sample is computed from these lists by means of a log-linear combination of these scores:

$$\hat{c} = \underset{1 \leqslant c \leqslant C}{\operatorname{argmax}}((1-\alpha)\log P(x_{\text{offline}}|c) + \alpha \log P(x_{\text{online}}|c)),$$

being C the number of word classes. The combination coefficients were estimated by exhaustive scanning of values over the biMod-IAM-PRHLT validation set. Table 45 shows a summary of the whole recognition engine.

Table 45: Bimodal recognition engine summary, including the combination coefficients. Table taken from [España-Boquera *et al.* 2010; Table 2].

Bimodal system	offline	online
# input features per frame	60	8
# HMM states	7	7
MLP hidden layers' size	192-128	192-128
MLP input context (left-current-right)	4-1-4	12-1-12
Combination coefficient	$(1-\alpha)=0.55$	$\alpha=0.45$

Table 46: Bimodal recognition engine performance (baseline from [Pastor *et al.* 2009; 2010]). Table taken from [España-Boquera *et al.* 2010; Table 3].

		Unimodal		Bimodal	
System		Offline	Online	Comb.	Rel. improv.
Val.	Baseline	27.6	6.6	4.0	39%
	HMM/ANN	12.7	2.9	1.9	34%
Test	HMM/ANN	12.7	3.7	1.5	59%

Experimental results

The ICPR contest organizers published baseline validation results in [Pastor *et al.* 2009]: Error rates were 27.6% for the offline modality and 6.6% for the online modality. A Naive Bayes combination of both unimodal systems resulted in a error rate of 4.0% for the bimodal task.

Our system achieved a 12.7% error rate for the offline validation set, and a 2.9% for the online validation set. Combining both sources resulted in a 1.9% validation error rate. The error rates for the test dataset were 12.7% for the offline data, 3.7% for the online data, and 1.5% for the bimodal task. Table 46 shows these results for validation and test together with baseline system for comparison purposes. The last column illustrates the relative improvement of the bimodal system over the online (best unimodal) system. As can be observed, close to 60% of improvement is achieved with the bimodal system w.r.t. using only the online system for the test set.

Our main interest in these experiments, here, is to illustrate the good performance of the online preprocessing of Section 14.8. Relating the contest [Pastor and Paredes 2010], our system differed with *the other*[17] in just 1 single test pattern. Best results were obtained by using matching instead of classification since all test labels were different and all test labels appeared in the test set.[18] We can make no great conclusions, we achieved the best performance in unimodal tests and all participants who performed classification differed in only 1 single test pattern (confidence intervals were not provided).

17 There were three participants, but *only two* recognition systems, since one of the teams used the results of the other in order to apply a matching technique that our research group considered but never applied assuming that this was not allowed.

18 We already wonder what is the utility of this assumption in practical scenarios and it is clear that matching is a much simpler task than the individual classification of each pattern. The idea of using matching came to our minds but we did not try believing this was unfair.

15.7 SUMMARY, CONCLUSIONS AND FUTURE WORK

This chapter has basically summarized some experimentation performed in HTR tasks using the preprocessing techniques described in Chapter 14 and the decoders proposed and implemented through this work. The focus has been on the assessment measures related to the quality of the recognition results rather than on the speed of decoders.

Most of the results reported in this chapter have been published elsewhere. The main preliminary results used to evaluate the preprocessing were published in [España-Boquera *et al.* 2011] although we have been able to improve them several times afterwards.

The aim of the first experimental part, described in Section 15.1 and in [España-Boquera *et al.* 2011], is twofold: 1) to show the validity of our preprocessing techniques, and 2) to study the suitability of hybrid HMM/ANN models for the "optical" part of HTR systems.[19]

The first claim has been empirically supported by comparing our preprocessing with a more traditional approach from the state-of-the-art [Pastor Gadea 2007]. Both HMM/GMMs and HMM/ANNs have been tried to this end. Relating the second claim, HMM/ANNs have systematically outperformed the HMM/GMM counterpart in both preprocessing techniques. An expected critique to this comparison if the lack of decorrelation techniques (such as PCA) in the feature extraction process. It is expected that HMM/GMMs are more sensible to that than HMM/ANNs because GMMs have used diagonal covariance matrices[20] whereas hybrid HMM/ANN models, for their part, do not need any of these assumptions and, indeed, are tolerant faced with irrelevant or redundant parameters. Also related to the feature extraction technique, it is worth mentioning that the most recent reported experiments, mainly due to Joan Pastor-Pellicer PhD work, avoids the use of feature extraction and relies on CNN layers to perform unsupervised feature extraction.

Another advantage of hybrid Markov models is the lower computational cost compared to HMM/GMMs, since a single forward evaluation of a unique MLP generates the values for all HMM states whereas a different Gaussian mixture is needed for each type of HMM state.[21]

The reader can observe that there is a delay of several years between the first and the last experiments. This may explain why some techniques, such as the use of a Gaussian noise perturbation for training optical models or the use of the MERT technique from [Och 2003] to better adjust GSF parameter, were not included from the very beginning.

19 The term "optical model" is a counterpart of the expression "acoustical model" from the ASR domain. We have used this expression in our papers although the nomenclature has not been coined and, indeed, other authors prefer terms such as "morphological models". To avoid confusions, we have avoided the use of any term when possible.

20 This kind of GMMs have been historically used in ASR with MFCC coefficients which are quite uncorrelated. Although some recent works on HTR apply these techniques, we tried to mimick the experimental conditions of previous works based on HMMs and on these features [Toselli *et al.* 2004; Pastor Gadea 2007].

21 Nevertheless, some remarks and comments on the evaluation of GMM can be found in Section 7.5.2.

In order to summarize, most of the recent improvements obtained since [España-Boquera *et al.* 2011] have been achieved in more or less straightforward ways (e.g. by increasing the lexicon size and augmenting the n-gram order). Other improvements have been achieved by fine-tuning some preprocessing and other HMM/ANN training related parameters.[22] Some of the more recent or noteworthy improvements have been obtained by using connectionist LM techniques (NN LMs) and by replacing the feature extraction by CNNs, as mentioned before.

We can also empirically observe that one of the most promising ways to improve the results is to combine different systems as described in [Zamora-Martínez *et al.* 2014] where HMM/ANNs and BLSTM/CTC systems, based on different preprocessing and feature extraction techniques were combined by means of ROVER.

Since our main proposals are related to decoding and to HTR preprocessing, we can roughly conclude that their suitability has been proven beyond reasonable doubt. Nevertheless, there is a lack of experimental results regarding the measure of decoding times, the effect of trying different pruning techniques or the comparisons between different decoding alternatives.

Future work

As a future work, we plan to move from text lines to entire paragraphs in order to be able to compare our new results with others that are being published at the paragraph level such as [Voigtlaender *et al.* 2015]. This recognition setting will not only benefit from the fact of using a proper LM (since paragraphs are made of sentences whereas lines are fragments which are recognized without context) but may also take profit from the fact that larger sequences may be more suited to include writer adaptation techniques.

We would also like to pursue with online recognition experiments in order to take profit of the code and techniques developed for this HTR modality whose experimental results has been limited so far to the isolated bimodal recognition despite of the fact that some preliminary experiments has been already conducted on the IAM-OnDB corpus [Liwicki and Bunke 2005] (see also Appendix A.2.2).

Another extension that we have planned as a future work is the extension of Section 15.5 (combining HMM/ANNs with holistic ANNs) from isolated to continuous HTR (lines and paragraphs) based on the ideas from Section 7.7. For this purpose, only the dataflow version of the two-stage decoder will be used (leaving aside one-pass decoders). More concretely, only the `seq_graph_gen` module described in Section 11.1 has to be adapted, while the DAG decoder may remain unchanged.

Many other ideas basically consists in performing more experiments with some preprocessing techniques, parameters and settings that cannot been carried out, for the moment, due to lack of time and computing power. Most of these ideas, such as the use of width normalization, has been described in Chapter 14.

22 For instance, by augmenting the input context and the sizes of hidden layers.

Relating the optical models, the training of HMM/ANNs can be ameliorated not only by improving the corresponding connectionist models used to estimate the posteriors (e.g. by using deeper models, by further exploting the idea of replacing the feature extraction pipeline by means of CNNs) but also by trying other HMM topologies in order to better support allographs or simply by maintaining the left-to-right topology while not forcing the same number of states for all graphemes. We also consider very promising the training at the sentence level instead of at the frame level, in the line of [Kingsbury 2009; Veselỳ *et al.* 2013]. We are willing to modify the training part of our April toolkit to this end.

Part IV

Conclusions and future work

16 CONCLUSIONS AND FUTURE WORK

NOT ONLY IS IT NOT RIGHT,
IT'S NOT EVEN WRONG!
Wolfgang Pauli

LIFE IS WHAT HAPPENS
WHILE YOU ARE BUSY
MAKING OTHER PLANS.
John Lennon

CONTENTS

This work deals with stochastic generative approaches to the joint segmentation and classification problem on sequences. It is focused on problems satisfying two properties: 1) they can be represented (at least approximately) in terms of one-dimensional sequences, and 2) solving these problems entails breaking the observed sequences down into non-overlapped segments which are associated to units taken from a finite repertoire (for instance, phonemes in speech) so that a sequential correspondence between the original signal and the underlying sequence of units is preserved (i.e. there is no reordering).

The segmentation and classification tasks usually required to determine the sequences of hidden units which best explain the observed signal are so intrinsically interrelated (fact known as "Sayre's Paradox") that they have to be performed jointly. Well known examples

Sayre's Paradox

of these problems include automatic speech recognition (ASR) and handwritten text recognition (HTR) which, despite being related to two-dimensional images, can be considered in some circumstances as fundamentally uni-dimensional. Examples of problems *not* covered by this framework include document layout analysis and non-monotone statistical machine translation.

Being a large field of research, it is impossible, in practice, to cover the broad spectrum of possibilities related with this particular topic. Although it would have been wiser to focus on a narrower topic "like a laser beam", we have opted to pursue the "successful trilogy" metaphor from [Aubert 2002] that was described in the introductory chapter. This metaphor states that, in order to obtain improvements in this mature field, the following fronts have to be jointly taken into account:

successful trilogy

1. a good formalization framework and powerful algorithms;
2. a clever design and implementation taking the best profit of hardware;
3. an adequate preprocessing and a careful tuning of all heuristics.

Besides, the structure of this report also reflects this trilogy by dividing the writing in three different parts, each one roughly related to the corresponding point of the trilogy:

1. problem statement and formalization;
2. proposed algorithmic solution;
3. experimentation and results.

Since we have to delimit our area of study, we have only focused on decoding techniques leaving aside other essential issues such as model estimation or model adaptation. Although decoding is a major issue in this work, other contributions (e.g. related to HTR preprocessing, language modeling and so on) are also provided.

Another particularity is the fact that the presentation of the state of the art and the presentation of novel proposals seem more intermixed than usually expected in a PhD report with a narrower scope where we can often observe a first part devoted to the state of the art and the novel proposals afterwards.

It would be a mistake to believe that the three parts of this work correspond, respectivelly to: state of the art, proposals and their experimental validation. But it would be a greater mistake, as detailed below, to believe that the contributions of this work are limited to the experimental results.

The rationale for mixing previous art with novel proposals throughout the writing is, as explained, a better presentation of ideas. This seem a consecuence of the broader focus, for better or worse. Prioritizing the clarity of exposition[1] over highlighting novelties has, as a counterpart, the requirement of clearly disentangling our contributions.

need to disentangle contributions

1 This work is evaluated once but, hopefully, could be readed for other purposes afterwards.

Another reason was, in our humble opinion (as developed in Section 1.3), the convenience of structuring what can be considered a mature field. Although probably the very idea of structuring can be called into question,[2] we believe that this effort, which we do not claim at all to have seriously carried out,[3] may have some a priori utility,[4] namely:

- it is a way to structure our own knowledge in the field in order not to reinvent the wheel;[5]
- to easily detect where new improvements can be done and, hopefully, to obtain them with less effort;
- to help other people interested in the field by grouping what is often dispersed in several papers;
- to provide a line of argument where contributions can be better contextualized;[6]
- because part of the intended contributions are not just novelties but also the way of organizing existing ideas in other arrangements, perhaps to promote or to influence readers towards certain points of views, some kind of "shameless lobbying". For instance: towards the use of segmental models, the exposition of HMMs with independence of the type of emission probabilities,[7] or to promote certain context free parsing technologies based on RTNs instead of resorting to CYK.[8]

16.1 BRIEF OVERVIEW HIGHLIGHTING OUR CONTRIBUTIONS (MY TWO CENTS)

Let us first briefly summarize the contents of each of the three parts in which this work has been divided. The purpose of this section is twofold since, on the one side, we hope that it will help the reader to better disentangle what, to the best of our knowledge, constitute the main[9] contributions (our two cents). On the other side, a brief overview might offer a panoramic view of this work (nevertheless, this has already been done in the road map of Section 1.4).

2 Rather than the idea itself, the fact that there are better contexts than a PhD for this.

3 Only sketched, we cannot pretend to bite off more than we can chew.

4 The following reasons seem an a posteriori justification after making it. This work has been written in a top-down way by creating a table of contents which has been filled afterwards. The truth is that we have not been able to effectively manage this procedure. This has borrowed a precious time that we could have better devoted to perform more experiments, to the point that we have finally had to remove some parts of the work (entire chapters, novel n-best algorithms, ASR experiments, etc.).

5 Or, more probably, to avoid claiming any novelty when rediscovering so many concepts which come naturally into mind when faced to the same requirements as the people who first made them, as would also happen to many other people.

6 I'm openly against making PhD reports as an agglomeration of published papers. This requires less effort but lacks a *line of argumentation* and lacks details.

7 For instance, by explicitly using the term HMM/GMM to refer to the use of Gaussian mixtures to model observed emissions instead of assuming them by default.

8 As an example, CYK requires CNF and the resulting parse trees are different from those of the original grammar. The RTN transform (based on 2NF) does not have this problem.

9 Detailing contributions of minor importance, scattered through the work, would have broken the spirit of a brief summary or overview.

16.1.1 Relating the first part

case studies

We have reviewed some case studies (Chapter 2) to provide examples that can help the reader to better understand the rest of the work. Unlike other reviews, we have avoided to discuss models and techniques at this point and we have just tried to point out the similarities underlying tasks that seem quite different on surface. This has led us to review different types of representation of the observed signal (frame, segment and landmark based observations) and to make some considerations about the relationship between preprocessing and the "pipeline problem", proposing some guidelines (summarized in Figure 20) that we have tried to follow: to allow preprocessing stages to delay ambiguities to later stages; to allow concepts belonging to later stages to influence previous ones; and to rely on statistical learning techniques instead of handcrafted heuristics. Although there are not relevant contributions in this chapter other than this vague proposal, these ideas have influenced the preprocessing techniques of Chapter 14.

problem hierarchy

We have tried to provide a classification of problem types (Chapter 3) that seems more exhaustive than others previously found in the literature. Nevertheless, it is restricted to the particular case of sequences explained by an underlying label sequence with a monotone correspondence between them. We have tried not to discuss problem solving techniques yet. The followed approach consisted first in characterizing problems by the type of input and output variables, but we have concluded that some cases are too complex to fit this scenario, specially when dealing with interactive systems where several actors (e.g. human supervisors, recognition systems) collaborate in an complex activity that cannot be established a priori (e.g. this activity may include the creation of more training data for improving previous models). We believe that this situation can be tackled by using ideas related to partial annotation and some proposals in this direction have been made. We have also had, in this chapter, the opportunity to review common input and output types (word graphs, confusion networks, etc.) as well as some assessment measures. There are not great contributions in this part either,[10] since the mentioned proposal of Section 3.3 has not been detailed enough to become such one, although we are very confident about their validity, if further developed.

models and, particularly, TSGMs

After discussing some problem types, it is now time to discuss how to solve them. To this end, a family of models called "two stage generative models" (TSGMs) is studied (Chapter 4). TSGMs are composed by two probabilistic stages and characterized by the interface between them (a sequence of hidden labels). This chapter starts by describing some preliminary concepts (no new contributions there although we are very proud of our review of graphical models (GMs) where we have resorted to a huge amount of bibliography until we have found

10 Less relevant contributions include a new type of oversegmentation information, the proposal of some new (albeit not very relevant) problem type (namely, the static analysis of the probability of accidental concatenations in keyword spotting) and a questionnaire "when faced with a new problem" that we have developed to summarize many features related to our problem types.

how to expose in a uniform way[11]). A hierarchy of TSGMs (relating their second stage) is proposed (including classes that have been left out of our scope) and some limitations, extensions or possible generalizations are studied. The relationship and some comparisons between TSGMs with other formalisms (graphical models, segmental models, and other ones such as the estimation of frame posteriors and CTC decoding) is also provided, were we have been specially critic with fixed dimension feature segments. This chapter is mostly devoted to prior art and to introduce the reader to the model type we are assuming, so there are no important contributions.

two-stage decoding

Solving the classical problems on TSGMs leads to the decoding technique known as two-stage where: 1) a DAG representing the possible segmentation of the observed signal together with the likelihoods of observing these signal segments given a (possibly contextualized) hidden label is obtained; and 2) the information about the hidden labels which may best explain the observed signal is computed from this DAG and from a language model (LM) that model the a priori probability of sequences of these labels.

one-pass decoding

A particular case, known as one-stage, deserves special attention. This happens when the TSGM can be factorized in such a way that *segmentation ambiguities disappear*: the most straightforward case occurs when HMMs are used so that the first part (the input DAG) only contains one-frame-length edges and the overall algorithm, *when LMs are regular*, becomes linear with the length of the observed signal. We believe that this point of view, that is not found in most works on the field, clarifies the relationship between one-pass and two-stage decoders. In any case, let us remember that the two-stage can deal with models that cannot be handled by the one-pass.

relationship

parsing & decoding

Parsing and decoding are addressed in Chapter 5 where we have basically restricted our study to weighted regular and context free models using semirings, language equations and the parsing as intersection approach. We have pointed out the interest on using recurrent transition networks (RTNs) as a normal form to represent CFGs and proposed a novel[12] algorithm for composing a FSA with a RTN that works fine even with null transitions and non-idempotent semirings. This has been achieved without resorting to the filter composition technique, which is related to the state-pair approach. Instead, we have proposed the homogeneous epsilon form for both FSA and RTNs and constructive algorithms to obtain them. We have also reviewed most classical parsing algorithms and discussed the additional factorization possibilities of our approach w.r.t. the more general (but not necessarily more suitable, in all cases) hypergraph framework. Relating contributions, this chapter concentrates the most relevant ones from the first part of this writing.

novel normal forms & composition algorithms

11 I have been nearly obsessed to find a connection between variable elimination, jointree and summary-product (a.k.a. belief propagation) algorithms. Surprisingly, I have not found it in the outstanding amount of literature on this matter, a notable exception being [Darwiche 2009] (although the exposition is focused on Bayesian networks and we have preferred to adapt the discourse to factor graphs). As with segment models, the detail and time devoted to GMs is not reflected by their actual use in this work, since we have finally had to shorten our expectations afterwards.

12 In the spirit of Bar-Hillel algorithm and similar ones properly cited in Chapter 5, but based on manipulating language equations and on the use of a novel normal form.

LMs

Chapter 6 describes language modeling, leaving aside some more engineering related aspects that are delayed to the second part (LM representation and interfaces). This review may also be considered unconventional since most works are exclusively focused on n-grams (that we prefer to call "count-based" to distinguish them from other n-grams types that include NN LMs and that we have called "feature-based"). Relating the novelties of this chapter, besides reviewing and organizing previous ideas,[13] our basic contribution has been to illustrate some less conventional uses of language modeling. A notable example has been to deal with imperfect transcription decoding that has been tackled up to now, and to the best of our knowledge, by explicitly adapting decoders (lack of time has prevented us from experimentally trying this idea). We can also mention the proposal of Section6.11 that goes along the lines of generalizing the lattice structure of [Dupont and Rosenfeld 1997]; unfortunately, as with the proposal of Section 3.3, the main problem, in our humble opinion, is not the idea itself but the fact that it has not been further developed in this work in order to take their full potential, being little more than a declaration of intent.

SMs

Chapter 7 reviews segment models (SMs). Discussing about SMs is relevant in our work since they offer a *raison d'être* for two-stage decoders and, for instance, one of the main differences between our two-stage recognizers (used in nearly all our experiments) and a more classical time-start decoder is their capability to use SMs other than HMMs. The proposed review is not a treatise on HMMs, but an attempt to offer an overall vision of the space of design. We have made a synthesis effort where SMs are mainly divided into implicit and explicit duration models, with a halfway category. We have also tried to provide a general classification of SMs and a scheme of frame emissions. On the negative side, these classifications are too coarse and the focus has been limited to model description, leaving aside estimation issues which are so crucial in practice. Another critic is that SMs have not finally had the importance we had assigned to them in the initial stages of this PhD. Nevertheless, we have started a very promising line of research by combining generative models with discriminative SMs (HMM/ANNs and holistic ANNs, respectively) and this has lead to the technique described in Section 7.7 and later validated in Section 15.5. This can be, perhaps, the main contribution of this chapter.

16.1.2 Relating the second part

The second part of the work correspond to the engineering and algorithmic aspects of the work. The main focus is on modular recognition systems based on the dynamic decoding approach and our desire to deal with TSGMs:

- modularity is addressed at several fronts: dataflow architecture and the use of components with a well defined API;
- dynamic decoders offer pros and cons w.r.t. static ones. We have opted for the dynamic approach basically because of their flexibility to combine certain particular models, as detailed through

13 Which is, at most, a doubtful second rate contribution.

the report. Nevertheless, we have to advise that the dichotomy between dynamic vs static is *misleading* since it does not cover the entire spectrum of decoding techniques (other possibilities, such as stack decoding, exist);

- in particular, several advantages of dynamic decoders are shown in the work, as is the suitability of the proposed dataflow architecture to construct recognition engines of the dynamic type, the possibility of using certain specialized algorithms for lexicon decoding, DAG generation algorithms that could even include the proposed SMs based on using discriminative holistic models at the work level (experimentally validated in Section 15.5) or how easy is to include NN LMs, to mention a few.

dataflow architecture

Let us now detail each of these points. An alternative to monolithic decoder implementations is the use of a dataflow architecture where, as shown in Chapter 8, several recognition systems can be constructed from a reduced repertoire of components (a metaphorical comparison is the use of lego pieces). We have put an special emphasis to show that these systems do not suffer the problems inherent to decoupled architectures (the use of feedback mechanisms is proposed to overcome these limitations, we are not aware of the use of this mechanism in previous literature). Besides, a novel DAG serialization protocol allows the proposed components to generate and to manipulate these DAGs while frames are being received so that the actual DAG, in an extreme case, do not need to be explicitly represented or stored in memory.

Relating contributions, although the use of a dataflow architecture is not novel by itself, we believe that it has not been used up to now to design decoders (well, some recognition systems internally use threads and queues and can be considered to some extent similar) and their main use in previous art seems relegated to be a wrapper to configure the signal processing part over an existing recognition engine, not at the core of the engine itself. Our system has been implemented as part of the April toolkit, is fully working and contains several components, based on a novel serialization protocol, that are described into more detail in subsequent chapters. A minor contribution has been to provide several examples of complete architectures based on them in order to show how to deal with multimodal, with interactive and multi-pass systems.

LM APIs & representation

Although language modeling was discussed in detail in the first part of this work (Chapter 6), it is now time to retake it from another point of view, paying attention to LM interfaces and to LM representations (Chapter 9). When dealing with LM interfaces, we have highlighted the capabilities for using pruning techniques and the novel proposal of a bunch mode that can be used by our decoders to improve efficiency and which is well suited for NN LMs where, no doubt, similar ideas has been applied.[14] The extension of the proposed LM API to the case of RTNs is also addressed.

Both count-based and feature-based n-grams are discussed from implementation and representation points of view, describing our representation for count-based models based on sorting states by their fan

14 In a quite ad hoc way, more coupled to decoders, rather than making a general API that we have also used in count based n-grams.

out, which is a quite naive but efficient way to speed up LM look ups and, to a less extent, to reduce space. This representation has also been extended to RTNs, paying special attention to make the format backwards compatible while being well suited for the novel RTN API. Other contributions of this chapter are related to NN LMs. Two main contributions are related to this part: on the one side, we address some techniques to speed up the evaluation of NN LMs. A quite weak contribution has been a refinement of the memoization of softmax normalization constants. This refinement has consisted in using a sole NN LM to model all lower order fallback models. A more relevant contribution has been a novel technique to estimate these normalization constants by using an auxiliary LM. Preliminary experiments are promising. Another contribution related to NN LMs is a novel type of NN LM whose internal tree lexicon structure perfectly reflects the tree lexicon, which allows LMLA for free. The implementation of the decoder adapted for this novel "LMLA NN LM" is one of our main priorities in our list of future work. A technique to speed up the initial layer of NN LMs, special suited for the case of lexicon free systems, although this technique dates from 2008 and has been published elsewhere as recently as 2014. A minor contribution of this chapter has been to show a "LM agnostic" technique to include LMs in an assisted transcription tool.

lexicon decoders

Besides LMs, another key component to construct decoders is the estimation of segments and their sequences based on a lexicon. Chapter 10 provides a throughout review of many algorithmic techniques associated to this issue and one of the first things we can observe from this chapter is their extension. Most of the techniques described here cannot be found in traditional books on the subject and are only scattered throughout several papers.[15] Although minor contributions are described (e.g. our proposal of jointly processing several frames is new, to the best of our knowledge), the main contribution of this chapter consist of several lexicon decoding algorithms. They have the particularity of taking profit of specialized lexicon model topologies to be able to perform decoding using an explicit representation of active states while avoiding the use of dictionary look-ups (e.g. hashing). Moreover, they are cache-friend and quite fast. Although preliminary experimental results where the speed of some of these decoders is compared with others from the state of the art, further experiments would have undoubtedly improved this chapter.

DAG generation

One of the main components of a dataflow recognition system is the DAG generator, which is discussed in Chapter 11 and which makes use of lexicon estimators from the previous chapter. Although this component can be easily described in terms of a transducer composition, the use of the proposed DAG serialization protocol, the inclusion of pruning techniques and the possibility of taking other features into account (e.g. information from an external over-segmenter, a control automaton to deal with initial and final optional silences/spaces or to deal with context dependent units, the possibility of clustering frame boundaries,...) makes it deserve an entire chapter. Besides this basic

15 Well, some of the techniques described here are novel and others are details provided by personal communication with other researchers, as is the case of one histogram pruning algorithm of David Llorens.

component, the construction of a graph from another one or the combination of graphs complements the exposition of graph generation in the dataflow framework.

We believe that most contents of this chapter are contributions since we are not aware of similar techniques in the literature excepting, as detailed in the chapter, some work related to the possibility of a modular implementation of time-conditioned recognition systems [Ortmanns *et al.* 1999]. The use of a control automaton provides a "poor man's" solution, for dynamic recognition systems, of a feature which is basically straightforward in the static composition approach. Another feature that is not found elsewhere is the capability of using external over-segmentation information. We have concluded that this feature is lacking from other systems basically because its usefulness is quite low in speech compared with what happens in handwriting. It suffices to match that with the fact that HTR decoding techniques have historically inherited the tradition of ASR to conclude that this feature, albeit useful, is not used nowadays in HTR systems other than ours. We would like to emphasize the fact that our proposed two-stage system based on this DAG generator would be more powerful than time conditioned systems unless HMMs and regular LMs (e.g. n-grams) are used and a simple example that has been nearly achieved (it suffices to connect the different already proven parts) is the use of the models tested in Section 15.5 to the case of continuous HTR. The DAG generation from a sequence has been much more developed and has been successfully used in our experiments while the construction of a word DAG from a subword one, or the combination of DAGs in multimodal scenarios, have not been experimentally tested for the moment.

DAG decoder

The other key component of a dataflow recognizer is the DAG decoder, which accounts for the second part of two-stage algorithm. This component makes use of LMs of Chapter 9 and has several similarities with the DAG generation of previous section, although an internal trellis data structure, instead of a pool of lexicon decoders, is now required. This trellis is used to obtain the list of n-best solutions, to extract a word graph or just the 1-best solution afterwards.[16]

novel one-pass decoder

One-pass decoders, which takes profit of specialized lexicon decoders and the use of bunch mode of the LM interface, are described. Two particularities are worth mentioning: firstly, they can extend the idea of avoiding dictionary look-ups to the entire system and, secondly, they can be parallelized in such a way that a thread is responsible of lexicon decoding, other of LM modeling and others are devoted to preprocessing and to the computation of frame emission likelihoods. A fully working system able to work at a load factor of 400% CPU in a 4-core machine has been obtained, which is not impressive by itself unless we also observe the speed-up w.r.t. a single thread execution. Although not yet tried, we believe that this recognizer could be further parallelized, when more cores were available, by splitting the lexicon decoders and the LM subsystem. Besides, the inclusion of over-segmentation techniques in one-pass decoders is described, which seems novel from what we have observed in the literature.

16 Due to lack of time, we have finally removed an entire chapter devoted to these different process where a novel n-best algorithm were proposed.

16.1.3 Relating the third part

This part is basically devoted to validate the ideas for NN LMs introduced in Chapter 9 and, mainly, to propose some novel HTR preprocessing techniques and their empirical validation using HMM/ANNs as optical models and the decoders proposed in this work.

Chapter 13 summarizes some language modeling experiments related to the techniques proposed in Chapter 9 such as the automaton representation, the use of a skipping NN LM as a fallback model for storing memoization constants or the estimation of these constants by means of an auxiliary LM.

Chapter 14 starts with a throughout review of HTR preprocessing stages starting from image cleaning and enhancing[17] (where the use of neural network filters has been proposed and used from the very beginning of this work [Hidalgo *et al.* 2005]) and including the common preprocessing stages briefly introduced in Section 2.7 but skipping layout analysis and text line extraction. Otherwise stated, we assume that we can depart from text image lines. The proposed preprocessing techniques can be characterized by using statistical pattern recognition techniques instead of geometrical handcrafted heuristics as was commonly done in more classical works. The extraction of interesting points from local extrema and their classification by means of ANNs is the distinguishing feature of our preprocessing. A noteworthy uncommon feature is the use of a non-uniform slant correction which is also based on ANNs. Other features, such as text width normalization, has not been tested for the moment. This offline text preprocessing has also been adapted to the online modality. Both modalities have been experimentally tested in the following chapter. Excepting the review of the classical state of the art alternative techniques, the entire chapter is a novel contribution of this PhD.

Chapter 15 contains the most experimental results of this work which have been basically restricted[18] to HTR. The main result is the empirical validation of preprocessing techniques from previous chapter as well as the validity of decoders implemented in our April toolkit and used in all these experiments. Both HMM/GMMs and HMM/ANNs and two different preprocessing techniques (ours and a more classical of [Pastor Gadea 2007] kindly[19] provided by the author) have been tested on the IAMdb task. This part corresponds to [España-Boquera *et al.* 2011]. Other experiments from Chapter 15 include a novel evaluation of IAMdb with some slightly improved HMM/ANNs by using NN LMs as language models, and their combination with a system based on BLSTMs with CTC decoding; some lexicon-free recognition experiments based on higher order character based NN LMs; experiments based on combining HMM/ANNs with discriminative holistic classifiers based on MLPs specialized for short words; and, finally, some bimodal experiments.

17 We have avoided the common use of binarization and have maintained images in what we have called "normalized gray level" where gray values are used for anti-aliasing purposes in the border of strokes.

18 In spite of the fact that we have also devoted a non-negligible effort in ASR (including the implementation of the feature extraction technique of [Nadeu *et al.* 1996] and the experimentation with the French Media and the Spanish TCstar corpora.

19 We are very grateful to Moisés for his generosity.

16.2 SOME CONCLUSIONS

If we had to summarize contributions in five lines, we would say: Novel RTN and transducer composition algorithms; novel dynamic decoding algorithms, including two-stage recognition systems able to deal with SMs other than HMMs; some LM related contributions; and a novel HTR preprocessing that has been successfully validated by using HMM/ANNs.

A general conclusion is that we have tried to follow the successful trilogy metaphor by tackling their three fronts into account and this has lead a bittersweet flavor: on the one side, we have developed fully working systems reaching very competitive results with them. On the other side, some of the contributions of each of the three parts remain somewhat unconnected between them so that the synergistic combination claimed in the beginning has not been tested in these cases. Other issues remain pending or not completely closed. We have the impression that just a slight additional effort would have help to obtain a more coherent product.

Let us now draw some conclusions on the different issues developed through this work.

Relating TSGMs, we have taken the old concept of two-stage decoding (used in the very beginnings of ASR for distance based systems) to describe a process that, although not new,[20] was unnecessary during the decades where ASR (and later HTR) were dominated by the same old same old "HMM and n-gram combination".[21] However, TSGMs are not limited to HMMs and this is the reason why SMs other than HMMs have been discussed. We have investigated DGMs and their decoding techniques and we should probably try to further relate these different families and systems and to have had the opportunity of tackling model estimation. Regarding SMs, we have not developed this area very much in this work but, at least, a novel SM type which is based on the combination of a generative model and a discriminative one (HMM/ANNs and holistic ANNs, respectively) has been experimentally tested. The down side is, of course, that it has not been used in unconstrained HTR for the moment in spite of the fact that the design was already done.

The novel transducer composition algorithms, extended to RTNs, are more elegant, in our humble point of view, than previous alternatives such as the filter composition technique required in the state-pair approach when using null transitions and non-idempotent semirings. However, they have the same worst case complexity and solve the very same problems. Besides, this alternative to filter composition requires a normal form transformation so, in principle, we cannot make great

20 For instance: some time-start decoding papers talk about "time-conditioned graphs" which are, essentially, equivalent to the DAGs used in two-stage decoders.

21 Let us remark that this "old combination" remains, nevertheless, in the spotlight. Consider hybrid HMMs based on deep (even recurrent) ANNs for emission estimation or the recent improvements on NN LMs, to mention two examples. Nevertheless, alternative formalisms exists, such as BLSTMs with CTC decoding or the current trend based on the use of deep learning techniques applied to what are generally known as end-to-end approaches.

conclusions about their practical relevance for the moment.[22] Nevertheless, we have shown how to tackle the parsing and decoding process in a very minimalist way (resorting to a minimal set of concepts).

With respect to decoding, some people may argue that it is just based on the same old same old dynamic programming (DP) approach although the novel specialized lexicon decoders and the two different types of recognition engines (one-pass and two-stage) have distinguishing and novel features. Both decoder types are ready to use an (optional) external over-segmenter, which may not only improve the decoding time but, even in some cases, the recognition results.

The one-pass decoder probably takes more profit of the specialized lexicon decoders that are harmoniously combined with the LMs API in bunch mode to obtain a multi-threaded engine able to take profit of multi-core architectures. We can conclude that it is more attractive, in practice, than the two-stage counterpart, which will be probably relegated to those cases where necessary: when using RTNs as LMs based on the memoization approach (when it is better than the dynamic expansion), when using SMs other than HMMs, or when the same input data is to be used simultaneously by several LMs since, in that case, the DAG generator is only run once. The conclusions are hence limited in this part since we have not been able to empirically show the advantages of SMs and RTNs as LMs.

Relating LMs, our automata representation, albeit naive, is practical and efficient, but probably not enough to replace other alternatives in other existing systems. The extension to RTNs, although designed, has not been tested in practice. The use of a skipping NN LM as a fallback model for storing memoization constants will be probably superseded by the estimation of these constants by means of an auxiliary LM, probably in the form of an auxiliary output of the same model, unless systems not requiring this normalization (e.g. by training procedures that reduce the variance of the softmax constant) achieve considerable improvements.

Relating HTR preprocessing, we can conclude that, although the major critique seems to be the need of a manual supervision, the effort is very reduced in practice (thanks to bootstrapping) and is worth the obtained improvements. It is possible further reduce this effort when moving to a new corpus since previous models can be used as a point of departure. This claim has been empirically validated by using not only the preprocessing models (the corresponding MLPs) of IAMdb but even the very same HMM/ANNs for other corpus (the French RIMES corpus [Augustin *et al.* 2006] in the ICDAR 2009 handwriting recognition competition, where we obtained the second position [Grosicki and El Abed 2009]). Also, Joan Pastor Pellicer, a member of our research group applying this preprocessing for historical documents, layout analysis and text line extraction has recently applied (see [Pastor-Pellicer *et al.* 2015b]) the same preprocessing models at least in Saint Gall [Fischer *et al.* 2011], Parzival [Fischer *et al.* 2009] and Esposalles [Romero *et al.* 2013] corpora for interest point classification.

22 Maybe a particular filter composition transducer is essentially equivalent to our technique. We have to further investigate these issues.

16.2.1 Some expected critiques

Let us try to advance some critiques which are expect to be found in this work. Of course, there are necessarily other critiques we cannot foresee.

The most expected critique, in our humble opinion, is the lack of a more exhaustive experimental validation and comparison of all the proposed algorithms, between them and with competing approaches. Not surprisingly, one of the effects of this critic is a huge amount of future work. Implementing and comparing all of them is a titanic effort even for a great research group working during a long period of time, not to mention our case. Indeed, what we have finally shown here could not have been possible without the invaluable help of working side by side with Pako Zamora, Jorge Gorbe and, in the last period, with Joan Pastor.

Relating a comparison with competing approaches, it is very difficult to make a fair one, which explains why we have shared lexicons and LMs with other research groups in some circumstances. We have also tried to highlight which things can be compared and to which extent. If comparing accuracies is difficult, dealing with decoding speed is even more problematic (although easier when comparing our own systems between them). Regarding other systems, a comparison between the static network representation approach and the dynamic approach we have favored in this work has not been performed.

Other critiques related to decoding are the following ones: 1) we have described and presented techniques to deal with context dependent units both in the lexicon models and across lexicon segments, but we have not performed experimental comparisons despite having described algorithms to do that. We can reply, in response, that hybrid HMM/ANNs can use a very wide neighborhood context at the frame level which allows us to obtain good context independent models (this fact has also been validated elsewhere in the case of ASR); 2) one of the advantages of two-stage decoders is the possibility of using more sophisticated modeling of segments, but we have not fully tried any of these models. We can only say that we have planned to extend our holistic model experiments form isolated to continuous HTR and lack of time have prevented to finally report them here; and 3) relating the claimed synergy of each part, some people would argue that the experimental part would have lead to similar results with other decoders (HTK [Young *et al.* 1993], Kaldi [Povey *et al.* 2011], etc.). To this respect, this critique reflects the fact that recognition results and decoding time should be taken jointly into account since faster decoders allow better results for the same computer resources (e.g. by allowing less aggressive pruning). The lack of this relationship in this report is another reasonable critique. Nevertheless, we believe that the use of over-segmenter information, that is certainly not present in the aforementioned alternative decoders, may slightly improve HTR results (we should have empirically validated that).

Relating the proposed preprocessing, one of the main critics (besides the need of manual supervision discussed before as well as in the conclusions of Chapter 14) is that we should have investigated the individual contribution of each preprocessing stage and that we should

have tried more settings instead of providing only a limited justification for the chosed ones.[23] The combinatorial explosion of experiments together with a limitation in resources (time and more computing facilities) is the *wild card* excuse for all these critics.

Other critics may be related to the PhD subject itself or to the extension form and structure of this report. We have tried to justify ourselves to this regard in the introduction (both in the introductory chapter and in the introduction of this chapter).

Let us finish these conclusions with some thoughts to this respect before entering the part of future work.

16.2.2 Some final thoughts

This work can be analyzed at several levels: We can put our attention to the details or we can try to obtain a panoramic point of view. Also, we can take for granted the premises given by the title and main objectives this work or, alternatively, adopt a "meta level" and question, even criticize, *the subject itself*.[24] The truth is that, for better or for worse, this work has not been carried out in the context of a particular PhD offer with a closed subject, which does not imply a complete freedom either. So lets us start by making a short conclusion about the subject: Although we are aware of several works on decoding, even PhD reports, we have not anticipated the explosion of bibliographic search. Despite that, we have only been able to touch a little portion of the space of design, focusing on dynamic decoders and leaving aside, due to lack of time, a reasonable comparison with other dynamic alternatives, with the static composition or even with stack decoding and other techniques. I now agree with others that transverse and quite abstract topics are not advised and that the focus has to be kept as narrower as possible. Perhaps the same subject would have been more manageable if performed in a research group with existing decoders instead of creating the entire recognition systems from scratch.

Our main issue in the initial steps was to take profit of the obvious capability of two-stage decoders to deal with models other than the same old, same old combination of HMMs and n-grams. We would also wanted to go beyond regular models and try to make use similar or more uniform techniques for regular and context free models, which lead us to be interested in RTNs since both expansion and memoization can be described as extensions of the classical Viterbi as usually applied on FSA.

Another problem is that we had to work in a practical context, where participating in projects and producing results implied trying to simultaneously fulfill a goal different from our initial PhD interests. So, metaphorically speaking: if we try to combine two "non-collinear vectors" we finish in a broader two-dimensional space. The astute reader might observe that this is reflected in this work: There are some trends towards SMs and GMs and some work on RTNs and parsing algo-

23 Sometimes based on prior art, others in quite preliminary experiments, others in statistical measures on training data and others (shamefully) due to lack of time and/or computing resources.

24 A known joke says that an engineer is someone who solves in a right way the wrong problem. Fortunately, I am not engineer but what in Spanish is known as *licenciado*.

rithms, while the experimental part is more related to language modeling (both count based n-grams and NN LMs), faster (specialized) decoders and HTR experiments based on a novel HTR preprocessing and on hybrid HMM/ANNs. The lack of uniformity between some theoretical aims and the experimental results remembers me the John Lennon quote at the beginning of this chapter.

Yes, there are results, some of them are competitive and novel, but the final structure and discourse is not perfectly closed due to the presence of pending issues, which leaves a bittersweet flavor. A more compact and unified work could have obtained in one of the three following ways: 1) by removing some theoretical results related to RTNs, SMs and decoders, whose main strength are not profited in the experiments; or 2) by proposing a more theoretical work without experiments; or, finally 3) by closing the gap between both trends. Indeed, we clearly supported the last option and nearly achieved that in the form of including the combination of discriminative and generative SMs in the DAG generator. Relating the use of RTNs in experimental tasks, we believe that they are not specially suited for transcription. Maybe, they could be used to perform some kind of concept tagging for SLU or simply use them to construct concepts richer than categories (e.g. dates and place descriptions suitable for a booking dialogue task [Bonneau-Maynard *et al.* 2005]).

need to close the gap

Besides discussing the PhD subject itself, we can also take into account the circumstances. Although it is not easy to assess a work without them, it is probably fairer. As mentioned below, it is quite easy to obtain experimental results in a short time when entering a research group with a consolidated infrastructure, prior experience on a subject and even lots of useful scripts. Having a concrete and narrower line to follow makes things probably easier and is perhaps the only reasonable way to fulfill a PhD work in the more common current case where students have only three years, albeit full-time, to complete it. Some of these PhDs have even an "industrial flavor" where PhD reports of the same group share their introduction (even the very same figures).[25] This is not the case here, for a lot of reasons, and this may warn those readers used to more usual PhD reports which are the reasonable way in habitual circumstances. This work has been carried out in a little research group without prior experience in HTR and all experiments have been carried out with a toolkit that we started from scratch[26] and developed during all these years by no more than three people (always partial time or, rather, full-time in other things and borrowed time to do that). I believe that the resulting April toolkit can be considered a more important result of this PhD work (in collaboration with the other authors, of course) than the published papers and that this report itself. It may seem now strange to implement a toolkit given the presence of so many alternatives,[27] but the circumstances where very different at the time of starting its development.

25 Tell the deed, not the doer.

26 More concretely, from an *emacs* scratch buffer.

27 For instance, Julia programming [Bezanson *et al.* 2012] has made impressive developments in libraries and toolkits in a few time. Other toolkits such as Kaldi [Povey *et al.* 2011] did not exists when our toolkit development started.

strongly biased towards teaching

This work has been done while working full-time as teacher and, although this was not at all unusual here and other people in these circumstances have successfully done a shorter report, this probably explains our clear bias towards writing something more akin to a introductory book than to a report of results. As stated in the preface, this report is more akin to the personal notes I would have liked to have had at the beginning than to the result to deliver.

16.3 FUTURE WORK

This section proposes some future lines that can be considered to pursue this work. Some of them have been anticipated and addressed in the corresponding "future work" sections of the preceding chapters. In order to show that the amount of future work is not related with the extension of this section, let us remark that, for a large amount, most of them can be summarized in:

> *Finishing the implementation of all algorithms and techniques described in the report and perform experimental comparison where the influence of each part is studied.*

Nevertheless, let us detail some possible future work extensions of PhD (including but not limited to finishing the aforementioned implementation of techniques described so far) divided by topic.

16.3.1 Related to decoding

April toolkit

Our most clear future work assignment related to decoding is to port the existing decoding algorithms from April toolkit to the April-ANN fork. This is a fork containing the ANN part (hence its name) that has been constantly improved, including many novel features (e.g. hyperparameter optimization, stacked denoising autoencoders, rectifier neural units (ReLU), CNNs, etc.), but it lacks many packages (including all our decoding algorithms, LMs, and the dataflow architecture). This port would allow us to remove unessential parts and to simplify some ones during the process. This would also allow us to use these decoders with better models (e.g. better hybrid HMM/ANNs and NN LMs). To summarize, two main decoding families are to be ported:

- the two-stage decoder based on the combination of the DAG generators of Chapter 11 and the DAG decoder of Chapter 12; and
- the multi-threaded one-pass decoder.

lexicon decoders

Let us remark that both of them make use of the same lexicon decoders and LM models, that should be also migrated. The most useful and efficient lexicon decoders are those based on the double ordering[28] of hypotheses, although we seek to implement and to experiment with other interesting proposals as is the case of jointly decoding

28 One-pass decoders use LM identifiers whereas two-stage decoders use time starts (frame frontier identifiers) but the algorithm use them in an agnostic way.

several frames (useful when decoders are used together with an over-segmenter). We also plan to try the OOV detection system proposed in Section 10.7 which are based on the parallel execution of a lexicon network and a lexicon-free decoders, which has some resemblances (but also differences) with [Kombrink *et al.* 2012].

improve two-stage decoder

The most obvious improvements relating the DAG generator are: 1) the clustering of frame boundaries, and 2) the inclusion of a module to combine generative with discriminative SMs.[29]

Relating the DAG decoder, we plan to make a version able to deal with the RTN extension of the LM interface of Section 9.3.3. It is worth mentioning that RTNs can be used in both one-pass and two-stage systems in a straightforward way by using a LM wrapper that is able to use the proposed RTN models as regular ones (offering the former LM API) by performing an on-the-fly model expansion. However, this procedure is probably not as efficient as the memoization version that would use the new RTN API. In particular, the resulting trellis in this last case allows the use of novel n-best extraction algorithms for context-free models. In any case, the main problem of RTNs, from our point of view, is not the lack of good decoders but the lack of good models able to provide some advantages over more traditional LMs (such as n-grams). We can only speculate, but our belief is that RTNs could find a niche in tasks other than transcription and more related to SLU (e.g. concept tagging) as mentioned before.

Also related to two-stage systems, and as pointed out in Chapter 8, it is easy to make a quite orthogonal improvement to the current dataflow architecture consisting in replacing the main loop that drives the execution of dataflow modules by a thread pool and by the use of an asynchronous event library such as `libuv`.[30] In order to improve the distributed capabilities of the architecture, we can also envisage the replacement of our quite ad hoc serialization of dataflow tokens by the use of libraries such as `nanomsg`.[31]

one-pass

Regarding the implemented one-pass recognition system, it is highly tuned for current multi-core architectures and has already been tested as a dictation tool[32] by emitting key events in Unix™/Linux machines using a quite rudimentary voice activity detection that should be improved in any case. As pointed out in Chapter 12, it is possible to further extend the parallelism capabilities of this decoder by splitting the lexicon decoder, and by further parallelizing the LM and the computation of scaled likelihoods (this part, in particular, is the most suitable for porting to GPUs). It is also possible to distribute the LM in a grid not only for performance reasons but also to provide a LM server suited for several decoder instances and to break the LMs into smaller parts, as pointed out in the conclusions of Chapter 12. The novel bunch mode extension of the LM API is particularly useful for NN LMs.

29 This extension could be probably tackled at the level of lexicon decoders.

30 `http://libuv.org/`

31 `http://nanomsg.org/`

32 Indeed, a practical (although not strictly related to research) result of this work could be to release a functional free dictation tool with pre-trained models. This depends on the availability of corpora with the adequate permissions to deliver the trained models. Collaboration with research groups from other Universities and countries would be desirable to this end.

LMLA NN LMs

Besides porting and adapting existing decoders, we would also want to implement other ones. Among them, we are specially interested in a particular one-pass decoder specially suited for the LMLA NN LMs described in Section 9.5.4. The extension of one-pass decoders for dealing with interactive transcription systems is also worth investigating. Most interactive transcription systems rely on the manipulation of a word graph obtained in a first pass, and we would like to investigate how not to be limited to the results obtained in a first and uninformed stage by applying all knowledge sources in a full search while taking profit as much as possible of the effort of previous recognition stages.

16.3.2 Related to language modeling

Several language modeling proposals have appeared in this work. The most obvious future work related to them is a greater experimental comparison of the different acceleration techniques for NN LMs described in Chapters 9 and 13, but other ideas are worth investigating. Centering our attention on NN LMs, we would also like to apply factored representations as well as the use of a cache in the line of [Zamora-Martinez *et al.* 2012].

Our novel LMLA NN LMs would also kill two birds with one stone: removing large softmax output and allowing a perfect LMLA to greatly enhance pruning (the only LMLA technique available in our current decoders is unigram smearing). As with other NN LMs, they can be enhanced with a cache input. Our interest in these models is at least to achieve state-of-the-art performance (e.g. in terms of PPL). We are specially interested in applying these techniques in ASR applications.

Although the idea of addressing OOVs detection from the lexicon decoders themselves is of our interest, we would also like to try ideas related with the combination of lexicon-free and lexicon-based LMs. Nevertheless, these ideas are now a hot topic which is being investigated at several research groups.

It is obvious that RTNs have constituted a relevant part of the first and more theoretically motivated part of this work. In order to take profit of these developments and also to better show the interest in this type of models, it is necessary to provide practical examples of tasks where RTNs may offer some advantage over more standard LMs. A possible application might be SLU. Translation is also a current topic where context free models are gaining attention, specially in the case of syntax-directed transduction although, in that case, we have to remark that synchronous CFGs are often used instead and, unfortunately, RTNs cannot be extended (at a first glance) to this case.

distributed LMs

Another future work project is to take advantage of the bunch mode at the LM interface to implement a distributed LM representation where a large LM is split into different computers [Brants *et al.* 2007]. The capability of managing large sets of LM and the division of the automata based LM by numerical properties of state identifier queries makes it easier to efficiently distribute queries between different computers and to merge the results afterwards (see Section 12.1.1).

16.3.3 Related to handwriting

preprocessing

The preprocessing method for offline handwriting proposed in this work can be improved in several ways as described in the conclusions of Chapter 14. Even without changing the preprocessing steps, a clear way to enhance it is to obtain better neural network classifiers. We do not mean only to retrain them with more and better labeled data, but rather to apply learning techniques, regularization methods, activation units and topologies (e.g. deep ones) not tried up to now. There is, for instance, a lot of non-labeled training data at our disposal that could be used to perform an unsupervised pre-training. Another obvious enhancement consists in reducing the computational cost. In particular, the classification of interest points and the non-uniform slant normalization are quite costly and it seems easy to accelerate them. Text width normalization (described in Section 14.5.2) is a preprocessing step that has been recently implemented but not tested yet.

One of the most interesting improvements that we plan to try is related to the text size normalization: instead of performing a linear vertical scaling of each of the three areas[33] in each image column, we want to perform a non-uniform scaling where there are not abrupt changes in the scale. The technique is quite easy to implement, but lack of time has prevented us to try this simple idea.

An interesting idea that we would like to try is to extend the ideas of [Varga and Bunke 2003] to our preprocessing. The idea consist in distorting text lines to increase the training data. We believe that our preprocessing is very suitable to apply these techniques in a very sophisticated way.

Some recent results proposed by Joan Pastor Pellicer [Pastor-Pellicer *et al.* 2014; 2015a] replace the detection and classification of interest points by other techniques which take all image columns into account, we would like to pursue our collaboration with this line of research.

recognition

Besides preprocessing, the main way to improve HTR is to obtain better optical models. Current models are based on HMM/ANNs using a left-to-right topology without skips and using the same number of states in all grapheme models. Better models can be obtained in several ways. Some preliminary results replacing the feature extraction used so far (based on [Toselli *et al.* 2004]) by CNNs are described[34] in Section 15.3 with impressive results. Other possible ways to enhance the HMM/ANNs include the use of better connectionist models used to estimate the posteriors (e.g. by using deeper models, even (bidirectional) recurrent models, or by further exploiting the idea of replacing the feature extraction pipeline by means of CNNs). In particular, we would like to adapt the technique of tied posteriors to this scenario. Of course, training the models at the sentence level instead of at the frame level, in the line of [Kingsbury 2009; Veselỳ *et al.* 2013], would probably lead to better results.

33 Namely: ascenders, main body area and descenders.

34 Let us remember that these results have been performed in collaboration with another member of our research group and that they are mostly part of his own PhD and discussed here because they benefit from the same preprocessing and decoders of this work.

Besides dealing with preprocessing, feature extraction and hybrid HMM/ANNs, we cannot forget in this work the use of segmental models other than HMMs. To this respect, we would like to apply CNNs to perform holistic classification of short words that have been scaled to a fixed size. This should improve our current results based on MLPs. However, CNNs combined with (bidirectional) recurrent networks to classify preprocessed segments is more general than scaling to a fixed size and would be extended to larger vocabularies. We plan to combine them with HMM/ANNs, as described in Section 7.7.

In any case, it is nearly mandatory to migrate from text line recognition to the recognition of entire pages when using IAMdb corpus in order to better compare with other recent works (e.g. [Voigtlaender *et al.* 2015]) and also to take better profit of LMs.

16.4 PUBLICATIONS

Although we have many contributions which have not been submitted yet, this work has already lead to some publications. The following list is divided into:

- works more closely related to this thesis;
- publications that are important for this thesis under the role of a collaboration author;
- other publications carried out during these years which are out of the main focus of this thesis.

16.4.1 Contributions derived from this thesis

Even though the publications derived from this thesis have already been listed and cited in the corresponding chapters, we include here a whole list according to their relevance.

Publications in JCR indexed journals

España-Boquera, S., Castro-Bleda, M., Gorbe-Moya, J. and Zamora-Martinez, F. (2011), «Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 33 (4), pp. 767–779 [España-Boquera *et al.* 2011]

Zamora-Martínez, F., España-Boquera, S., Castro-Bleda, M. and Palacios-Corella, A. (2015), «Fall Back Skipping NNLM: Efficient Neural Network Language Models by Precomputation and Stochastic Training», *Speech Communication (submitted to)*, pp. – [Zamora-Martínez *et al.* 2015]

Publications in CORE indexed conferences

España, S., Castro, M. J. and Hidalgo, J. L. (2004), «The SPARTACUS-Database: a Spanish Sentence Database for Offline Handwriting Recognition», in «Proceedings of the International

Conference on Language Resources and Evaluation (LREC)», vol. I, pp. 227–230 CORE C. [España *et al.* 2004]

Zamora-Martínez, F., Castro-Bleda, M., España-Boquera, S. and Gorbe-Moya, J. (2010b), «Unconstrained Offline Handwriting Recognition using Connectionist Character N-grams», in «International Joint Conference on Neural Networks (IJCNN)», pp. 4136–4142 CORE A. [Zamora-Martínez *et al.* 2010b]

España-Boquera, S., Gorbe-Moya, J., Zamora-Martínez, F. and Castro-Bleda, M. J. (2010), «Hybrid HMM/ANN models for bimodal online and offline cursive word recognition», in «Recognizing Patterns in Signals, Speech, Images and Videos», pp. 14–21 CORE B. [España-Boquera *et al.* 2010]

Non indexed journals and conferences

Hidalgo, J. L., España, S., Castro, M. J. and Pérez, J. A. (2005), «Enhancement and cleaning of handwritten data by using neural networks», in «Pattern Recognition and Image Analysis», vol. 3522 of *Lecture Notes in Computer Science*, pp. 376–383 International Conference. [Hidalgo *et al.* 2005]

España-Boquera, S., Gorbe-Moya, J. and Zamora-Martínez, F. (2007), «Semiring Lattice Parsing Applied to CYK», in «Pattern Recognition and Image Analysis, IbPRIA», pp. 603–610, Springer International Conference. [España-Boquera *et al.* 2007]

Espana-Boquera, S., Zamora-Martínez, F., Castro-Bleda, M. and Gorbe-Moya, J. (2007), «Efficient BP algorithms for general feedforward neural networks», in «Bio-inspired Modeling of Cognitive Tasks», pp. 327–336, Springer International Conference. [Espana-Boquera *et al.* 2007]

España-Boquera, S., Castro-Bleda, M. J., Zamora-Martínez, F. and Gorbe-Moya, J. (2007), «Efficient Viterbi algorithms for lexical tree based models», in «Advances in Nonlinear Speech Processing», pp. 179–187, Springer International Conference. [España-Boquera *et al.* 2007]

Gorbe-Moya, J., España-Boquera, S., Zamora-Martínez, F. and Castro-Bleda, M. J. (2008), «Handwritten Text Normalization by using Local Extrema Classification», in «International Workshop on Pattern Recognition in Information Systems», pp. 164–172 International Conference. [Gorbe-Moya *et al.* 2008]

Zamora-Martínez, F., Castro-Bleda, M. J., España-Boquera, S. and Gorbe, J. (2010), «Improving isolated handwritten word recognition using a specialized classifier for short words», in «Proceedings of the Current topics in artificial intelligence, and 13th conference on Spanish association for artificial intelligence», CAEPIA'09, pp. 61–70 National Conference. [Zamora-Martínez *et al.* 2010]

Zamora-Martínez, F., Castro-Bleda, M., España-Boquera, S. and Gorbe, J. (2010a), «Improving Isolated Handwritten Word Recognition Using a Specialized Classifier for Short Words», in «Current Topics in Artificial Intelligence», vol. 5988 of *Lecture Notes in Computer Science*, pp. 61–70 National Conference. [Zamora-Martínez *et al.* 2010a]

Palacios-Corella, A., Zamora-Martínez, F., España-Boquera, S. and Castro-Bleda, M. (2014), «First steps towards Skipping NNLMs», in «VIII Jornadas en Tecnología del Habla and IV Iberian SLTech Workshop (IberSPEECH)», pp. 111–118 [Palacios-Corella *et al.* 2014]

16.4.2 Collaborations with other authors

This section contains publications that are important for this thesis under the role of a collaboration author.

Publications in JCR indexed journals

Zamora-Martínez, F., Frinken, V., España-Boquera, S., Castro-Bleda, M., Fischer, A. and Bunke, H. (2014), «Neural network language models for off-line handwriting recognition», *Pattern Recognition*, vol. 47 (4), pp. 1642–1652 [Zamora-Martínez *et al.* 2014]

Publications in CORE indexed conferences

Zamora-Martínez, F., Castro-Bleda, M. and España-Boquera, S. (2009a), «Fast Evaluation of Connectionist Language Models», in «International Work-Conference on Artificial Neural Networks», vol. 5517 of *Lecture Notes in Computer Science*, pp. 33–40, Springer CORE B. [Zamora-Martínez *et al.* 2009a]

Castro-Bleda, M. J., España, S., Gorbe, J., Zamora, F., Llorens, D., Marzal, A., Prat, F. and Vilar, J. (2009b), «Improving a DTW-based recognition engine for on-line handwritten characters by using MLPs», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 1260–1264, IEEE CORE A. [Castro-Bleda *et al.* 2009b]

Pastor-Pellicer, J., Zamora-Martínez, F., España-Boquera, S. and Castro-Bleda, M.-J. (2013), «F-Measure as the Error Function to Train Neural Networks», in «Advances in Computational Intelligence», vol. 7902 of *Lecture Notes in Computer Science*, pp. 376–384, Springer CORE B. [Pastor-Pellicer *et al.* 2013]

Pastor-Pellicer, J., España-Boquera, S., Zamora-Martínez, F., Afzal, M. Z. and Castro-Bleda, M. J. (2015c), «Insights on the Use of Convolutional Neural Networks for Document Image Binarization», in «Advances in Computational Intelligence», pp. 115–126, Springer International Publishing CORE B. [Pastor-Pellicer *et al.* 2015c]

Pastor-Pellicer, J., España-Boquera, S., Castro-Bleda, M. J. and Zamora-Martínez, F. (2015a), «A combined Convolutional Neural Network and Dynamic Programming approach for text line normalization», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. – [Pastor-Pellicer *et al.* 2015a]

Non indexed journals and conferences

Gorbe-Moya, J. (2005), *Preproceso, parametrización y reconocimiento de escritura manuscrita mediante técnicas geométricas y estadísticas*, Master's thesis, Facultad de Informàtica. Universidad Politécnica de Valencia, supervised by Salvador España [Gorbe-Moya 2005]

Castro, M. J., Díaz, W., Ferri, F. J., Ruiz-Pinales, J., Jaime-Rivas, R., Blat, F., España, S., Aibar, P., Grau, S. and Griol, D. (2005), «A holistic classification system for check amounts based on neural networks with rejection», in «Proc. Int. Conf. on Pattern Recognition and Machine Intelligence», vol. 3776 of *Lecture Notes in Computer Science*, pp. 310–314 International Conference. [Castro *et al.* 2005]

Zamora, F., España, S., Gorbe, J. and Castro, M. J. (2006), «Entrenamiento de Modelos de Lenguaje Conexionistas con Grandes Vocabularios», in «IV Jornadas en Tecnologia del Habla», pp. 141–144 [Zamora *et al.* 2006]

Llorens, D., Prat, F., Marzal, A., Vilar, J., Castro, M., Amengual, J., Barrachina, S., Castellanos, A., Espana, S., Gómez, J., Gorbe, J., Gordo, A., Palazón, V., Peris, G., Ramos-Garijo, R. and Zamora, F. (2008), «The UJIpenchars Database: A Pen-Based Database of Isolated Handwritten Characters», in «Proceedings of the International Conference on Language Resources and Evaluation (LREC)»European Language Resources Association (ELRA) CORE C. [Llorens *et al.* 2008]

Castro-Bleda, M. J., España-Boquera, S. and Zamora-Martínez, F. (2009a), *Encyclopedia of Artificial Intelligence*, chap. Behaviour-based Clustering of Neural Networks, pp. 144–151 [Castro-Bleda *et al.* 2009a]

Zamora-Martínez, F., Castro-Bleda, M. J., España-Boquera, S. and Gorbe-Moya, J. (2009), «Mejoras del reconocimiento de palabras manuscritas aisladas mediante un clasificador específico para palabras cortas», in «Conferencia de la Asociación Española para la Inteligencia Artificial», pp. 539–548 [Zamora-Martínez *et al.* 2009]

Martínez-Morató, J. (2011), *Aplicación para la supervisión en el preproceso de imágenes para escritura manuscrita*, Master's thesis, Facultad de Informàtica. Universidad Politècnica de Valencia, supervised by Salvador España and Francisco Zamora [Martínez-Morató 2011]

Zamora-Martinez, F., España-Boquera, S., Castro-Bleda, M. J. and De-Mori, R. (2012), «Cache Neural Network Language Models Based on Long-Distance Dependencies for a Spoken Dialog System», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 4993–4996 International Conference. [Zamora-Martinez *et al.* 2012]

Pastor-Pellicer, J., España-Boquera, S., Zamora-Martínez, F. and Castro-Bleda, M. J. (2014), «Handwriting Normalization by Zone Estimation using HMM/ANNs», *Frontiers in Handwriting Recognition (ICFHR), International Conference on* Internationally recognized. [Pastor-Pellicer *et al.* 2014]

16.4.3 Other publications

During these years some contributions in other related research fields, out of the main focus of this thesis, were done.

Publications in CORE indexed conferences

Zamora-Martínez, F., España-Boquera, S. and Castro-Bleda, M. J. (2007), «Behaviour-based clustering of neural networks applied to document enhancement», in «IWANN'07: Proceedings of the 9th international work conference on Artificial neural networks», pp. 144–151 CORE B. [Zamora-Martínez *et al.* 2007]

Khalilov, M., Fonollosa, J. A. R., Zamora-Martínez, F., Castro-Bleda, M. J. and España-Boquera, S. (2008a), «Arabic-English translation improvement by target-side neural network language modeling», in (ELRA), E. L. R. A., ed., «Proceedings of the International Conference on Language Resources and Evaluation (LREC)» CORE C. [Khalilov *et al.* 2008a]

Khalilov, M., Fonollosa, J., Zamora-Martinez, F., Castro-Bleda, M. and España-Boquera, S. (2008b), «Neural Network Language Models for Translation with Limited Data», in «Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on», vol. 2, pp. 445–451 CORE B. [Khalilov *et al.* 2008b]

Vilar, J. M., Castro-Bleda, M. J., Zamora-Martínez, F., España-Boquera, S., Gordo, A., Llorens, D., Marzal, A., Prat, F. and Gorbe-Moya, J. (2010), «A Flexible System for Document Processing and Text Transcription», in «Current Topics in Artificial Intelligence», vol. 5988 of *LNCS*, pp. 291–300 CORE B. [Vilar *et al.* 2010]

Frinken, V., Zamora-Martinez, F., Espana-Boquera, S., Castro-Bleda, M. J., Fischer, A. and Bunke, H. (2012a), «Long-short term memory neural networks language modeling for handwriting recognition», in «International Conference on Pattern Recognition (ICPR)», pp. 701–704, IEEE CORE B. [Frinken *et al.* 2012a]

Non indexed journals and conferences

Castro, M. J., España, S., Marzal, A. and Salvador, I. (2001c), «Transcriptor ortográfico-fonético para el castellano», *Procesamiento del Lenguaje Natural*, vol. 27, pp. 241–245, iSSN: 1135–5948 [Castro *et al.* 2001c]

Castro, M. J., España, S., Marzal, A. and Salvador, I. (2001b), «Grapheme-to-phoneme conversion for the Spanish language», in «Pattern Recognition and Image Analysis. Proceedings of the IX Spanish Symposium on Pattern Recognition and Image Analysis», pp. 397–402, AERFAI National Conference. [Castro *et al.* 2001b]

Castro-Bleda, M. J., Díaz-Villanueva, W., Domínguez-Rubio, J. L. and España-Boquera, S. (2002), «Prediction and Discrimination of Pharmacological Activity by Using Committees of Artificial Neural Networks», in «Ensemble Methods for Learning Machines. Proc. of International School on Neural Nets "E. R. Caianiello"» International Conference. [Castro-Bleda *et al.* 2002]

Zamora-Martínez, F., Castro-Bleda, M., Tortajada-Velert, S. and España-Boquera, S. (2009b), «A Connectionist approach to Part-Of-Speech Tagging», in «International Conference on Neural Computation», pp. 421–426 International Conference. [Zamora-Martínez *et al.* 2009b]

Vilar, J. M., Castro-Bleda, M. J., Zamora-Martínez, F., España-Boquera, S., Gordo, A., Llorens, D., Marzal, A., Prat, F. and Gorbe-Moya, J. (2009), «State: A Flexible System for Document Processing and Text Transcription», in «Conferencia de la Asociación Española para la Inteligencia Artificial», pp. 467–476 [Vilar *et al.* 2009]

Zamora-Martínez, F., Castro-Bleda, M., Tortajada-Velert, S. and España-Boquera, S. (2009c), «Adding morphological information to a connectionist Part-Of-Speech tagger», in «Current Topics in Artificial Intelligence», vol. 5988 of *Lecture Notes in Computer Science*, pp. 191–200 National Conference. [Zamora-Martínez *et al.* 2009c]

Castro, M. J., España, S., Llorens, D., Marzal, A., Prat, F., Vilar, J. M. and Zamora, F. (2011), «Speech interaction in a multimodal tool for handwritten text transcription», in «ICMI», pp. 299–302 International Conference. [Castro *et al.* 2011]

Khalilov, M., Fonollosa, J. A., Castro Bleda, M. J., España Boquera, S. *et al.* (2013), «Neural network language models to select the best translation», *Computational Linguistics in the Netherlands Journal*, (3), pp. 217–233 International Journal. [Khalilov *et al.* 2013]

Castro-Bleda, M., España, S., Prat, F., Vilar, J., Llorens, D., Marzal, A. and Zamora, F. (2014), «Human or Computer Assisted Interactive Transcription: Automated Text Recognition,

Text Annotation, and Scholarly Edition in the Twenty-First Century», *Mirabilia*, pp. 0247–253 International Journal. [Castro-Bleda *et al.* 2014]

Part V

Appendix

A CORPORA

CONTENTS

DATABASES play an important role in the development, evaluation and improvement of language models and recognition systems.

It is important to acknowledge that, in order to properly compare different systems, it does not suffice that they are evaluated on the same corpora but also that they use the same partitions for training, validation and test and, when it makes sense, they make use of the same lexicon or dictionaries and language models.

This chapter briefly summarizes some databases or corpora used in this work.

A.1 TEXT

Text corpora are useful for training and evaluating LMs and for extracting lexicons. They have sometimes been useful during the construction of some HTR and ASR tasks. As an example, the IAMdb handwritten database described in Section A.2.1 has been constructed by copying text fragments from the LOB corpus described in Section A.1.1. Reciprocally, some text corpora are based on speech or handwritten transcriptions.

A.1.1 Lancaster Olso Bergen (LOB)

The Lancaster Oslo Bergen Corpus [Johansson *et al.* 1986] is an English text corpus composed of three different corpora. The practical relevance of LOB corpus in this work is mainly due to the fact that it has been used to construct IAMdb database [Marti and Bunke 2002] described in Section A.2.1 Some statistics of the corpora are shown in Table 47.

Table 47: Statistics for LOB corpus.

Corpora	# Sentences	# Words	# Lexicon
LOB (excluding IAM Val & Test)	47.7K	1.06M	57.7K
Brown	56.6K	1.14M	55.5K
Wellington	58.3K	1.14M	54.4K
Total	162.6K	3.34M	103.4K

A.1.2 One Billion Word Benchmark

The "One billion word benchmark" [Chelba *et al.* 2013] is a corpus for evaluating statistical language modeling techniques. Its name emphasizes the fact that it comprises almost one billion words of training data obtained from the "EMNLP 2011 Sixth Workshop on Statistical Machine Translation", which is considered a large dataset at today's standards. It is publicly available[1] and, besides the corpus, we can also download several scripts as well as the PPL performance of lots of LM types and their combination.

A.2 HANDWRITING

As explained in Section 2.7, handwritten text recognition can be considered as part of a broader field known as "document recognition" which also covers layout analysis and is not limited to handwritten but also to typewritten or mixed handwritten-typewritten documents.

Relating handwritten documents, we can also distinguish at least two different input modalities (offline and online) and different levels of difficulty (from isolated characters to continuous handwriting, also explained in Section 2.7). Even for a given modality, we can consider other factors such as the way of performing the acquisition: we can distinguish between writing on a paper or on a whiteboard. It is not the same to write freely or to copy a text, writing a postal address or a bank check. It may also be important the language (for instance, French or Spanish may have accents not present in English), etc. Similar observations can be made in the case of speech (e.g. spontaneous vs read).

Several databases are available depending on each of these factors. Among them, the best known ones are CEDAR [Hull 1994], UNIPEN [Guyon *et al.* 1994], IRONOFF [Viard-Gaudin *et al.* 1999], IAMDB [Marti and Bunke 2002], IAMBonDB [Liwicki and Bunke 2005] and RIMES [Augustin *et al.* 2006]. Other less known databases include UJIpenchars [Llorens *et al.* 2008] (online isolated characters), Spartacus [España *et al.* 2004] (Spanish text lines), etc.

1 https://code.google.com/p/1-billion-word-language-modeling-benchmark

Table 48: Off-line IAMdb statistics.

Dataset	# Lines	# Words	# Lexicon
Train	6,161	53,871	7,750
Val	920	8,672	2,425
Test	2,781	25,424	5,314
Total	9,862	87,967	11,320

A.2.1 IAMdb

IAM database [Marti and Bunke 1999; 2002] is a database of handwritten text in the offline modality comprising scanned forms that some writers have performed by copying a text without restrictions on the writing style or the writing instrument used. An example is shown in Figure 215.

The version 3.0 of this database includes over 1 500 scanned forms from more than 650 different writers. These forms correspond to a total of more than 13 000 fully transcribed handwritten lines. The sentences have been extracted from the Lancaster-Oslo/Bergen (LOB) text corpus [Johansson *et al.* 1986] described in Section A.1.1.

The corpus is labeled in such a way that experiments can be performed at different levels: page/paragraph, sentences, lines and isolated words, as detailed in Section 15.1.1. The major part of experiments that we have conducted correspond to the text line recognition task.

The subset and partitioning of the IAM database used in this work consists of 6 161 training lines (from 283 writers), 920 validation lines (56 writers) and 2 781 test lines (161 writers). All these data sets are disjoint, and no writer has contributed to more than one set. These partitions are the same as those used in several works by Bunke et al. [Bertolami and Bunke 2008b; Graves *et al.* 2009] and differs slightly from the task defined at the IAM web page[2]. Training is the same in both partitions, but our test set includes more lines.

Statistics of this database can be seen in Table 48. A total of 87 967 instances of 11 320 distinct words occur in the union of the training, validation, and test sets. Lexicon is modeled with 79 characters shown in Table 24: 26 lowercase letters, 26 uppercase letters, 10 digits, 15 punctuation marks, the space, and a character for garbage symbols.

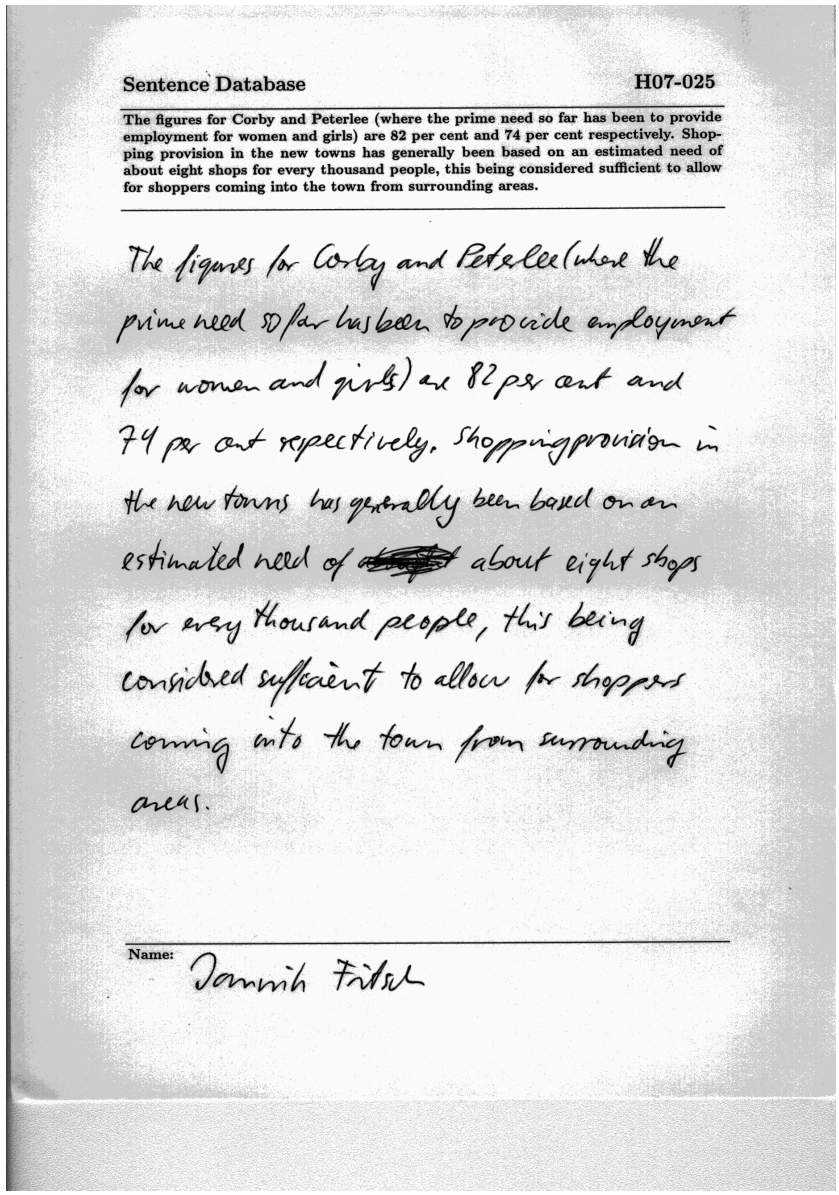
A.2.2 IAM-OnDB

IAM-OnDB [Liwicki and Bunke 2005] is a publicly available[3] large online handwritten sentences database. As with their offline IAMdb counterpart, texts included in this database have been taken from the LOB corpus [Johansson *et al.* 1986] described in Section A.1.1.

The text has been acquired on a whiteboard, as illustrated in Figure 216.

2 The IAM database is available for public download along with a task definition at: `http://www.iam.unibe.ch/fki/databases/iam-handwriting-database`.

3 `http://www.iam.unibe.ch/fki/databases/iam-on-line-handwriting-database`



Sentence Database H07-025

The figures for Corby and Peterlee (where the prime need so far has been to provide employment for women and girls) are 82 per cent and 74 per cent respectively. Shopping provision in the new towns has generally been based on an estimated need of about eight shops for every thousand people, this being considered sufficient to allow for shoppers coming into the town from surrounding areas.

The figures for Corby and Peterlee (where the prime need so far has been to provide employment for women and girls) are 82 per cent and 74 per cent respectively. Shopping provision in the new towns has generally been based on an estimated need of about eight shops for every thousand people, this being considered sufficient to allow for shoppers coming into the town from surrounding areas.

Name: Jannik Fritsch

Figure 215: Example of form from IAMdb.

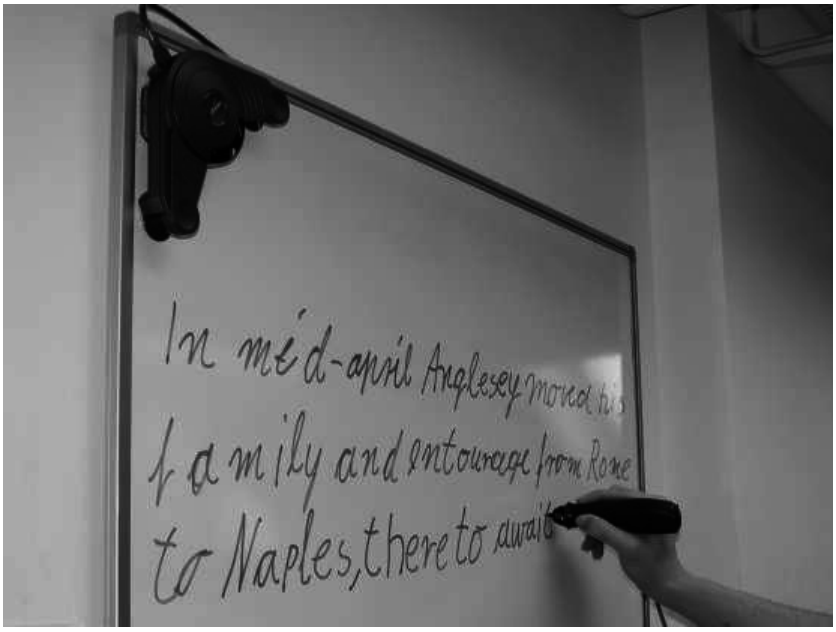


Figure 216: Whiteboard recording during the acquisition of IAM-OnDB. Observe the acquisition device in the left upper corner of the whiteboard. Figure taken from [Liwicki and Bunke 2005; Fig. 1].

A.2.3 biMod-IAM-PRHLT

The biMod-IAM-PRHLT corpus is a bimodal dataset of online and offline isolated handwritten words [Pastor *et al.* 2009], publicly available for academic research. It is composed of 519 handwritten word classes with several online and offline word instances extracted from the publicly available IAM corpora [Marti and Bunke 2002; Liwicki and Bunke 2005]. Figure 217 shows some examples of bimodal samples from the corpus.

The writers of the online and offline samples are generally different. The offline samples are presented as grey-level images, and the online samples are sequences of coordinates describing the trajectory of an electronic pen.

The corpus is partitioned into training and validation for common benchmarking purposes. A test partition was held-out and it was used for the "Bi-modal Handwritten Text Recognition" contest organized during the 2010 ICPR [Pastor and Paredes 2010]. Both validation and test partitions are composed of a bimodal sample (a pair of an online and an offline instance) for each word in the vocabulary. Some basic statistics of this corpus are shown in Table 49.

A.2.4 RIMES

RIMES project is the acronym of "Reconnaissance[4] et Indexation de données Manuscrites et de facsimilES". The database associated

4 In English: "Recognition and indexing of handwritten documents and faxes".

Table 49: Basic statistics of the biMod-IAM-PRHLT corpus and their standard partitions. The vocabulary is composed of 519 word classes.

Running words	Online	Offline
Training set	8 342	14 409
Validation set	519	519
(Hidden) test set	519	519
Total	9 380	15 447

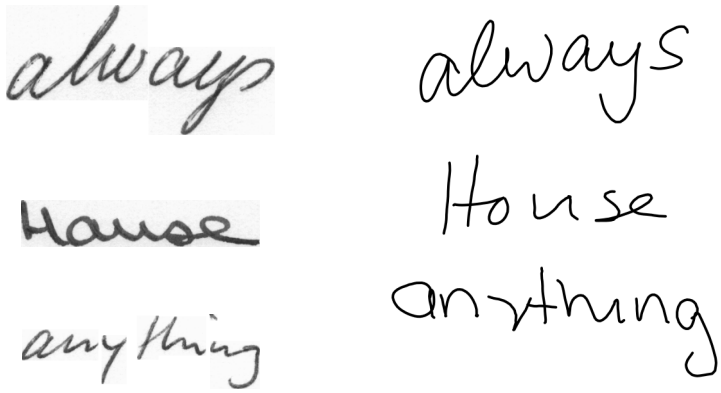


Figure 217: Examples of word samples from the bimodal corpus: left, offline images, and right, online samples.

to the RIMES evaluation campaign for handwritten mail processing [Augustin *et al.* 2006] Some examples are shown in Figure 218. The use of this database in our work has been limited to our participation in the ICDAR 2009 contest.

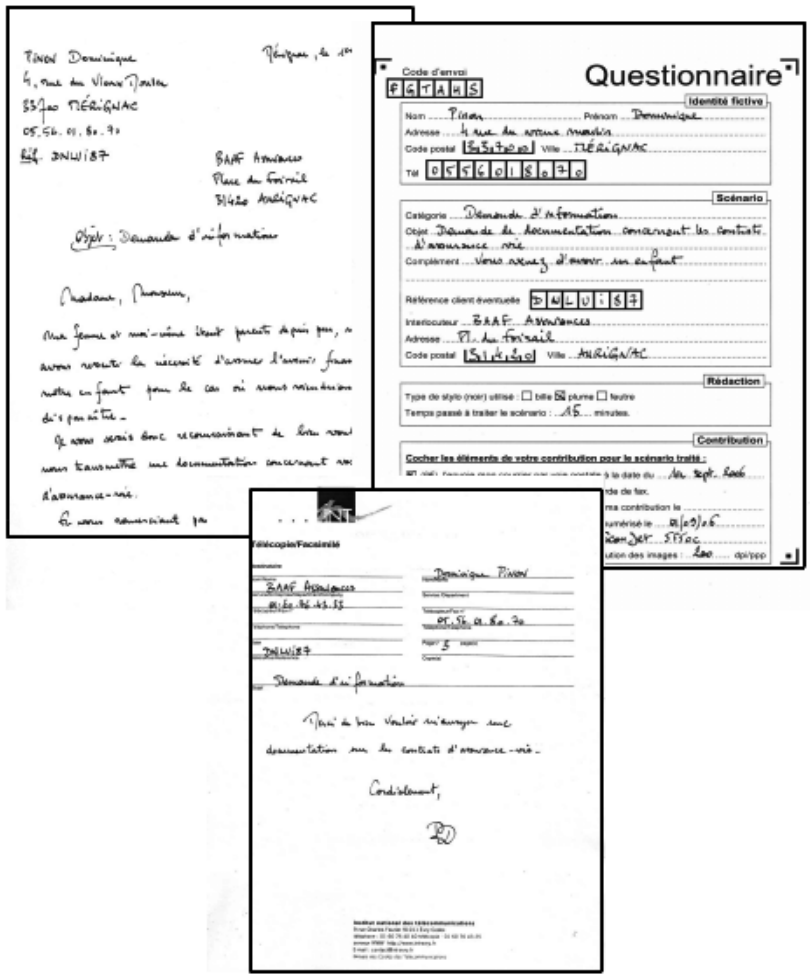


Figure 218: Examples from RIMES database.

BIBLIOGRAPHY

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J. *et al.* (2015), «TensorFlow: Large-scale machine learning on heterogeneous systems, 2015», *Software available from tensorflow. org*. (Cited on page 325.)

Abdou, S. and Scordilis, M. S. (2004), «Beam search pruning in speech recognition using a posterior probability-based confidence measure», *Speech Communication*, vol. 42 (3), pp. 409–428. (Cited on page 8.)

Abney, S. (1991), «Parsing By Chunks», in «Principle-Based Parsing», vol. 44 of *Studies in Linguistics and Philosophy*, pp. 257–278, Springer Netherlands. (Cited on page 29.)

Abram, G. and Treinish, L. (1995), «An Extended Data-Flow Architecture for Data Analysis and Visualization», in «Proceedings of the 6th conference on Visualization '95», p. 263, IEEE. (Cited on page 325.)

Agam, G. and Dinstein, I. (1997), «Geometric Separation of Partially Overlapping Nonrigid Objects Applied to Automatic Chromosome Classification», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 19 (11), pp. 1212–1222. (Cited on page 3.)

Aho, A. and Ullman, J. (1972), *The theory of parsing, translation, and compiling*, Prentice-Hall, Inc. (Cited on pages 183 and 192.)

Aho, A. V., Sethi, R. and Ullman, J. D. (1986), *Compilers, Principles, Techniques*, Addison wesley. (Cited on pages 225 and 228.)

Aibar, P. (1997), *Diseño de un decodificador acústico-fonético mediante una aproximación basada en distancias*, Ph.d. thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, advisor: Dr. Francisco Casacuberta Nolla. (Cited on pages 131 and 163.)

Aibar, P., Marzal, A., Vidal, E. and Casacuberta, F. (1992), «Problems and Algorithms in Optimal Linguistic Decoding: A Unified Formulation», in «International Conference on Spoken Language Processing (ICSLP)», pp. 1625–1628. (Cited on page 165.)

Aji, S. and McEliece, R. (2000), «The generalized distributive law», *Information Theory, IEEE Transactions on*, vol. 46 (2), pp. 325–343. (Cited on page 115.)

Al-Bashabsheh, A. (2014), *Normal Factor Graphs*, Ph.D. thesis, University of Ottawa. (Cited on page 123.)

Alabau, V., Sanchis, A. and Casacuberta, F. (2012), «On the optimal decision rule for sequential interactive structured prediction», *Pattern Recognition Letters*, vol. 33 (16), pp. 2226–2231. (Cited on pages 8 and 83.)

Allauzen, C., Mohri, M. and Roark, B. (2003), «Generalized algorithms for constructing statistical language models», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», pp. 40–47. (Cited on page 359.)

Allauzen, C., Riley, M. and Schalkwyk, J. (2011), «A Filter-based Algorithm for Efficient Composition of Finite-State Transducers», *International Journal of Foundations of Computer Science*, vol. 22 (08), pp. 1781–1795. (Cited on page 205.)

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W. and Mohri, M. (2007), «OpenFst: a general and efficient weighted finite-state transducer library», in «Implementation and Application of Automata», pp. 11–23, Springer. (Cited on pages 346 and 360.)

Alleva, F., Huang, X. and Hwang, M. (1996), «Improvements on the Pronunciation Prefix Tree Search Organization», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», p. 133, IEEE. (Cited on pages 427, 430, and 516.)

Alleva, F., Hon, H., Huang, X., Hwang, M., Rosenfeld, R. and Weide, R. (1992), «Applying SPHINX-II to the DARPA Wall Street Journal CSR task», in «Proceedings of the workshop on Speech and Natural Language», HLT '91, pp. 393–398. (Cited on page 47.)

Alleva, F., Huang, X., Hwang, M.-Y. and Jiang, L. (1998), «Can continuous speech recognizers handle isolated speech?», *Speech Communication*, vol. 26 (3), pp. 183–189. (Cited on pages 4 and 54.)

Almazan, J., Gordo, A., Fornes, A. and Valveny, E. (2014), «Word Spotting and Recognition with Embedded Attributes», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 36 (12), pp. 2552–2566. (Cited on page 164.)

Amatriain, X. (2007), «CLAM: A framework for audio and music application development», *Software, IEEE*, vol. 24 (1), pp. 82–85. (Cited on page 325.)

Amengual, J. and Vidal, E. (1998), «Efficient error-correcting Viterbi parsing», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 20 (10), pp. 1109 –1116. (Cited on pages 407 and 470.)

Andreou, A., Kamm, T. and Cohen, J. (1994), «Experiments in vocal tract normalization», in «Proc. CAIP Workshop: Frontiers in Speech Recognition II», p. 1994. (Cited on page 47.)

Antoniol, G., Brugnara, F., Cettolo, M. and Federico, M. (1995), «Language model representations for beam-search decoding», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 588 –591 vol.1. (Cited on page 411.)

Aradilla, G., Vepa, J. and Bourlard, H. (2005), «Improving speech recognition using a data-driven approach», Tech. rep., IDIAP. (Cited on page 163.)

Aradilla, G., Vepa, J. and Bourlard, H. (2007), «An Acoustic Model Based on Kullback-Leibler Divergence for Posterior Features», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 4, pp. 657–660. (Cited on pages 308 and 311.)

Arisoy, E., Chen, S. F., Ramabhadran, B. and Sethy, A. (2013), «Converting neural network language models into back-off language models for efficient decoding in automatic speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 8242–8246. (Cited on pages 261 and 369.)

Arnborg, S. (1985), «Efficient algorithms for combinatorial problems on graphs with bounded decomposability—A survey», *BIT Numerical Mathematics*, vol. 25 (1), pp. 1–23. (Cited on page 123.)

Artieres, T., Marukatat, S. and Gallinari, P. (2007), «Online Handwritten Shape Recognition Using Segmental Hidden Markov Models», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 29 (2), pp. 205–217. (Cited on page 290.)

Astrov, S. (2007), *Optimization of algorithms for large vocabulary isolated word recognition in embedded devices*, Ph.D. thesis, Technische Universität München, Munich, Germany. (Cited on pages 412 and 422.)

Aubert, L., Woods, R., Fischaber, S. and Veitch, R. (2013), «Optimization of Weighted Finite State Transducer for Speech Recognition», *Computers, IEEE Transactions on*, vol. 62 (8), pp. 1607–1615. (Cited on pages 405, 418, and 421.)

Aubert, X. (1989), «Fast look-ahead pruning strategies in continuous speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 659 –662 vol.1. (Cited on page 422.)

Aubert, X. L. (2002), «An overview of decoding techniques for large vocabulary continuous speech recognition», *Computer Speech and Language*, vol. 16, pp. 89–114. (Cited on pages 11, 14, 99, 415, 427, 509, and 618.)

Augustin, E., Carré, M., Grosicki, E., Brodin, J.-M., Geoffrois, E. and Prêteux, F. (2006), «Rimes evaluation campaign for handwritten mail processing», in «International Workshop on Frontiers in Handwriting Recognition (IWFHR)», pp. 231–235. (Cited on pages 604, 628, 646, and 650.)

Austin, S., Schwartz, R. and Placeway, P. (1991), «The forward-backward search algorithm», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 697–700 vol. 1. (Cited on page 425.)

Bacchiani, M., Riley, M., Roark, B. and Sproat, R. (2006), «MAP adaptation of stochastic grammars», *Computer Speech and Language*, vol. 20 (1), pp. 41–68. (Cited on page 258.)

Bahl, L., Brown, P., de Souza, P. and Picheny, M. (1988), «Acoustic Markov models used in the Tangora speech recognition system», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 497–500 vol.1. (Cited on page 298.)

Bahl, L., Brown, P., de Souza, P. and Mercer, R. (1989a), «A tree-based statistical language model for natural language speech recognition», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37 (7), pp. 1001–1008. (Cited on page 267.)

Bahl, L., Bellegarda, J., deSouza, P., Gopalakrishnan, P., Nahamoo, D. and Picheny, M. (1991), «A new class of fenonic Markov word models for large vocabulary continuous speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 177–180. (Cited on page 45.)

Bahl, L. R., Jelinek, F. and Mercer, R. L. (1983), «A Maximum Likelihood Approach to Continuous Speech Recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 5 (2), pp. 179–190. (Cited on pages 132 and 272.)

Bahl, L. R. *et al.* (1989b), «Large vocabulary natural language continuous speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 465–467. (Cited on pages 44 and 604.)

Baird, H. and Thompson, K. (1990), «Reading chess», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 12 (6), pp. 552–559. (Cited on page 268.)

Baker, J. (1975), «The DRAGON system–An overview», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 23 (1), pp. 24–29. (Cited on page 286.)

Bakis, R. (1976), «Continuous speech recognition via centisecond acoustic states», *The Journal of the Acoustical Society of America*, vol. 59, p. 97. (Cited on page 296.)

Bar-Haim, R., Sima'an, K. and Winter, Y. (2005), «Choosing an optimal architecture for segmentation and POS-tagging of modern Hebrew», in «Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages», pp. 39–46. (Cited on page 29.)

Bar-Hillel, Y., Perles, M. and Shamir, E. (1961), «On Formal Properties of Simple Phrase Structure Grammars», *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, vol. 14, pp. 143–172, reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on their Theory and Application*, Addison-Wesley 1964, 116–150. (Cited on pages 175, 186, 194, 210, 231, 232, and 236.)

Barber, D. (2012), *Bayesian reasoning and machine learning*, Cambridge University Press. (Cited on page 106.)

Barrault, L., Servan, C., Matrouf, D., Linares, G. and De Mori, R. (2008), «Frame-based acoustic feature integration for speech understanding», in «Acoustics, Speech, and Signal Processing, IEEE

International Conference on (ICASSP)», pp. 4997 –5000. (Cited on page 82.)

Bauer, L. (1993), «Manual of Information to Accompany The Wellington Corpus of Written New Zealand English», Tech. rep., Department of Linguistics, Victoria University, Wellington, New Zealand. (Cited on page 581.)

Baumann, T., Atterer, M. and Schlangen, D. (2009), «Assessing and improving the performance of speech recognition for incremental systems», in «Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics», NAACL '09, pp. 380–388. (Cited on page 102.)

Bax, E. (2005), «Checkpoint method for choice recovery in dynamic programming», *Bulletin of Mathematical Biology*, vol. 67, pp. 719–736. (Cited on page 508.)

Bayer, A. O. and Riccardi, G. (2012), «Joint Language Models for Automatic Speech Recognition and Understanding», in «Spoken Language Technology Workshop (SLT), 2012 IEEE», pp. 199–203. (Cited on page 251.)

Bazillon, T. (2011), *Transcription et traitement manuel de la parole spontanée pour sa reconnaissance automatique*, Ph.D. thesis, Laboratoire d'Informatique de l'Université du Maine. (Cited on page 83.)

Bazzi, I., Schwartz, R. and Makhoul, J. (1999), «An omnifont open-vocabulary OCR system for English and Arabic», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 21 (6), pp. 495 –504. (Cited on page 564.)

Bellegarda, J. (2004), «Statistical language model adaptation: review and perspectives», *Speech Communication*, vol. 42 (1), pp. 93–108. (Cited on pages 244 and 259.)

Bellegarda, J. R. (2000), «Exploiting latent semantic information in statistical language modeling», *Proceedings of the IEEE*, vol. 88 (8), pp. 1279–1296. (Cited on page 267.)

Bellman, R. E. (1957), *Dynamic Programming*, Princeton University Press. (Cited on page 233.)

Bengio, S. and Heigold, G. (2014), «Word embeddings for speech recognition», in «Conference of the International Speech Communication Association (Interspeech)», pp. 1053—1057. (Cited on page 164.)

Bengio, Y., De Mori, R., Flammia, G. and Kompe, R. (1991), «Global optimization of a neural network-hidden Markov model hybrid», in «International Joint Conference on Neural Networks (IJCNN)», vol. ii, pp. 789 –794 vol.2. (Cited on page 308.)

Bengio, Y., LeCun, Y., Nohl, C. and Burges, C. (1995), «LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition», *Neural Computation*, vol. 7 (6), pp. 1289–1303. (Cited on page 62.)

Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C. (2003), «A Neural Probabilistic Language Model», *Journal of Machine Learning Research*, vol. 3 (2), pp. 1137–1155. (Cited on pages 255, 263, and 369.)

Benzeghiba, M., Mori, R. D., Deroo, O., Dupont, S., Erbes, T., Jouvet, D., Fissore, L., Laface, P., Mertins, A., Ris, C., Rose, R., Tyagi, V. and Wellekens, C. (2007), «Automatic speech recognition and speech variability: A review», *Speech Communication*, vol. 49 (10-11), pp. 763–786. (Cited on page 42.)

Berstel, J., Boasson, L., Carton, O. and Fagnot, I. (2010), «Minimization of automata», *arXiv preprint arXiv:1010.5318*. (Cited on page 414.)

Bertolami, R. and Bunke, H. (2008a), «Ensemble methods to improve the performance of an English handwritten text line recognizer», in «Arabic and Chinese Handwriting Recognition», pp. 265–277, Springer. (Cited on page 580.)

Bertolami, R. and Bunke, H. (2008b), «Hidden Markov models-based ensemble methods for offline handwritten text line recognition», *Pattern Recognition*, vol. 41 (11), pp. 3452–3460. (Cited on pages 580, 596, and 647.)

Bertoldi, N., Federico, M., Falavigna, D. and Gerosa, M. (2008), «Fast speech decoding through phone confusion networks.», in «Conference of the International Speech Communication Association (Interspeech)», pp. 2094–2097. (Cited on pages 394 and 506.)

Beyerlein, P., Ullrich, M. and Wilcox, P. (1997), «Modelling and decoding of crossword context dependent phones in the Philips large vocabulary continuous speech recognition system», in «European Conference on Speech Communication and Technology (Eurospeech)», vol. 3, pp. 1163–1166. (Cited on page 417.)

Bezanson, J., Karpinski, S., Shah, V. B. and Edelman, A. (2012), «Julia: A Fast Dynamic Language for Technical Computing», *Computing Research Repository (CoRR)*, vol. abs/1209.5145. (Cited on pages 324 and 631.)

Bharath, A. and Madhvanath, S. (2011), «HMM-based Lexicon-driven and Lexicon-free Word Recognition for Online Handwritten Indic Scripts», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 34 (4), pp. 670–682. (Cited on page 54.)

Bianne-Bernard, A.-L., Menasri, F., Mohamad, R.-H., Mokbel, C., Kermorvant, C. and Likforman-Sulem, L. (2011), «Dynamic and Contextual Information in HMM Modeling for Handwritten Word Recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 33 (10), pp. 2066 –2080. (Cited on page 151.)

Billot, S. and Lang, B. (1989), «The structure of shared forests in ambiguous parsing», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», pp. 143–151. (Cited on pages 27, 53, and 230.)

Bilmes, J. (2010), «Dynamic graphical models», *Signal Processing Magazine, IEEE*, vol. 27 (6), pp. 29–42. (Cited on page 148.)

Bilmes, J. and Bartels, C. (2005), «Graphical Model Architectures for Speech Recognition», *Signal Processing Magazine, IEEE*, vol. 22 (5), pp. 89–100. (Cited on pages 147 and 148.)

Bilmes, J. and Zweig, G. (2002), «The graphical models toolkit: An open source software system for speech and time-series processing», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 4, pp. 3916–3919. (Cited on page 148.)

Bilmes, J. A. (2003), «Buried Markov models: a graphical-modeling approach to automatic speech recognition», *Computer Speech and Language*, vol. 17 (2–3), pp. 213–231. (Cited on page 290.)

Bilmes, J. A. (2004), «Graphical models and automatic speech recognition», in «Mathematical foundations of speech and language processing», pp. 191–245, Springer. (Cited on page 149.)

Bilmes, J. A. (2006), «What HMMs Can Do», *IEICE - Transactions on Information and Systems*, vol. E89-D (3), pp. 869–891. (Cited on page 288.)

Bilmes, J. A. and Kirchhoff, K. (2003), «Factored language models and generalized parallel backoff», in «Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume of the Proceedings of HLT-NAACL 2003–short papers - Volume 2», pp. 4–6. (Cited on pages 256, 280, 355, and 371.)

Bird, S. and Liberman, M. (2000), «A Formal Framework for Linguistic Annotation (revised version)», *Computing Research Repository (CoRR)*, vol. cs.CL/0010033. (Cited on page 89.)

Bisani, M. and Ney, H. (2002), «Investigations on joint-multigram models for grapheme-to-phoneme conversion», in «International Conference on Spoken Language Processing (ICSLP)», pp. 105–108. (Cited on page 44.)

Bisani, M. and Ney, H. (2004), «Bootstrap estimates for confidence intervals in ASR performance evaluation», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 409–412. (Cited on page 595.)

Bisani, M. and Ney, H. (2005), «Open Vocabulary Speech Recognition with Flat Hybrid Models», in «Conference of the International Speech Communication Association (Interspeech)», pp. 725–728. (Cited on pages 269, 270, and 599.)

Bishop, C. (2006), *Pattern recognition and machine learning*, springer New York. (Cited on pages 52, 106, 114, 288, and 307.)

Bishop, C. M. (1995), *Neural networks for pattern recognition*, Oxford University Press. (Cited on page 543.)

Blanc, O. (2006), *Algorithmes d'analyse syntaxique par grammaires lexicalisées : optimisation et traitement de l'ambiguité*, Ph.D. thesis, Université de Marne la Vallée. (Cited on page 255.)

Blanc, O. and Constant, M. (2006), «Outilex, a linguistic platform for text processing», in «Proceedings of the COLING/ACL on Interactive presentation sessions», pp. 73–76. (Cited on page 189.)

Blanc, O., Constant, M. *et al.* (2005), «Lexicalization of grammars with parameterized graphs», *RANLP, Borovets (Bulgaria)*, pp. 117–121. (Cited on page 188.)

Bolanos, D. (2012), «The Bavieca open-source speech recognition toolkit», in «Spoken Language Technology Workshop (SLT), 2012 IEEE», pp. 354–359. (Cited on page 323.)

Bonafonte, A. and Mariño, J. (1996), «Language modeling using x-grams», in «International Conference on Spoken Language Processing (ICSLP)», vol. 1, pp. 394–397. (Cited on pages 266 and 345.)

Bonneau-Maynard, H., Rosset, S., Ayache, C., Kuhn, A. and Mostefa, D. (2005), «Semantic annotation of the french media dialog corpus», in «Conference of the International Speech Communication Association (Interspeech)», pp. 3457–3460. (Cited on pages 12 and 631.)

Bordel, G., Varona, A. and Torres, M. (1997), «K-TLSS(S) language models for speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 2, pp. 819 –822 vol.2. (Cited on pages 265, 345, and 359.)

Bottou, L., Bengio, Y. and Le Cun, Y. (1997), «Global training of document processing systems using graph transformer networks», in «Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on», pp. 489 –494. (Cited on page 160.)

Bottou, L. and LeCun, Y. (2005), «Graph Transformer Networks for Image Recognition», *Bulletin of the International Statistical Institute (ISI)*, 55th Session. (Cited on pages 154 and 160.)

Bourlard, H. and Dupont, S. (1997), «Subband-based speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 2, pp. 1251 –1254 vol.2. (Cited on pages 81, 138, and 425.)

Bourlard, H., Hermansky, H. and Morgan, N. (1996), «Towards increasing speech recognition error rates», *Speech Communication*, vol. 18 (3), pp. 205–231. (Cited on pages 15, 96, and 286.)

Bourlard, H. and Morgan, N. (1994), *Connectionist speech recognition—A hybrid approach*, vol. 247 of *Engineering and computer science*, Kluwer Academic. (Cited on pages 156, 307, and 308.)

Bozinovic, R. M. and Srihari, S. N. (1989), «Off-Line Cursive Script Word Recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 11 (1), pp. 68–83. (Cited on pages 544, 552, and 553.)

Brakensiek, A., Rottland, J., Kosmala, A. and Rigoll, G. (2000), «Off-Line Handwriting Recognition Using Various Hybrid Modeling Techniques And Character N-Grams», in «International Workshop on Frontiers in Handwriting Recognition (IWFHR)», pp. 343–352. (Cited on pages 270, 565, and 599.)

Brakensiek A, R. G. (2001), «A Comparison of Character N-Grams and Dictionaries Used for Script Recognition», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 241–245. (Cited on pages 270 and 599.)

Brants, T., Popat, A., Xu, P., Och, F. and Dean, J. (2007), «Large language models in machine translation», in «Proc. of Empirical Methods in Natural Language Processing (EMNLP)», pp. 858–867. (Cited on pages 344, 383, and 634.)

Breuel, T. M. (2008), «The OCRopus open source OCR system», in «Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series», vol. 6815, p. 14. (Cited on page 346.)

Bridle, J., Brown, M. and Chamberlain, R. (1982), «An algorithm for connected word recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 7, pp. 899–902. (Cited on page 7.)

Bridle, J. S. (2004), «Towards better understanding of the model implied by the use of dynamic features in HMMs», in «Conference of the International Speech Communication Association (Interspeech)», pp. 725–728. (Cited on pages 5, 136, 290, and 319.)

Brown, P., Spohrer, J., Hochschild, P. and Baker, J. (1982), «Partial traceback and dynamic programming», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 7, pp. 1629–1632. (Cited on pages 9 and 508.)

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D. and Mercer, R. L. (1993), «The mathematics of statistical machine translation: parameter estimation», *Comput. Linguist.*, vol. 19, pp. 263–311. (Cited on pages 31 and 67.)

Brugnara, F. (2003), «Context-dependent search in a context-independent network», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 360–363. (Cited on page 399.)

Brugnara, F. and Federico, M. (1997), «Dynamic Language Models for Interactive Speech Applications», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 1827–1830. (Cited on pages 231 and 265.)

Bunke, H. (2003), «Recognition of Cursive Roman Handwriting – Past, Present and Future», in «International Conference on Document Analysis and Recognition (ICDAR)», vol. 1, pp. 448–459. (Cited on page 579.)

Burget, L. (2004), *Complementarity of speech recognition systems and system combination*, Ph.D. thesis, Faculty of Information Technology, Brno University of Technology. (Cited on page 102.)

Burr, D. J. (1982), «A normalizing transform for cursive script recognition», in «International Conference on Pattern Recognition (ICPR)», pp. 1027–1030. (Cited on pages 544 and 551.)

Caesar, T., Gloger, J. and Mandler, E. (1993), «Preprocessing and feature extraction for a handwriting recognition system», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 408–411. (Cited on page 553.)

Caesar, T., Gloger, J. and Mandler, E. (1995), «Estimating the baseline for written material», in «International Conference on Document Analysis and Recognition (ICDAR)», vol. 1, pp. 382–385. (Cited on pages 546 and 547.)

Cai, M.-Q., Ling, Z.-H. and Dai, L.-R. (2015), «Statistical parametric speech synthesis using a hidden trajectory model», *Speech Communication*, vol. 72, pp. 149–159. (Cited on page 302.)

Callison-Burch, C., Koehn, P., Monz, C., Schroeder, J. and Fordyce, C. S., eds. (2008), *Proceedings of the Third Workshop on Statistical Machine Translation*. (Cited on page 531.)

Callison-Burch, C., Koehn, P., Monz, C., Peterson, K. and Zaidan, O., eds. (2010), *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*. (Cited on pages 531 and 533.)

Callison-Burch, C., Koehn, P., Monz, C., Peterson, K. and Zaidan, O., eds. (2011), *Proceedings of the Sixth Workshop on Statistical Machine Translation*. (Cited on pages 12 and 528.)

Câmpeanu, C., Santean, N. and Yu, S. (2001), «Minimal cover-automata for finite languages», *Theoretical Computer Science*, vol. 267 (1), pp. 3–16. (Cited on page 414.)

Cardenal-Lopez, A., Diguez-Tirado, F. and Garcia-Mateo, C. (2002), «Fast LM look-ahead for large vocabulary continuous speech recognition using perfect hashing», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 705–708 vol.1. (Cited on pages 357, 427, and 431.)

Casacuberta, F. and de la Higuera, C. (1999), «Optimal linguistic decoding is a difficult computational problem», *Pattern Recogn. Lett.*, vol. 20 (8), pp. 813–821. (Cited on pages 6, 145, 162, 173, and 234.)

Casacuberta, F. and Vidal, E. (2004), «Machine Translation with Inferred Stochastic Finite-State Transducers», *Computational Linguistics*, vol. 30, pp. 205–225. (Cited on pages 32 and 266.)

Casacuberta, F., Vidal, E. and Picó, D. (2005), «Inference of finite-state transducers from regular languages», *Pattern Recognition*, vol. 38 (9), pp. 1431–1443. (Cited on page 269.)

Casacuberta, F., Ney, H., Och, F., Vidal, E., Vilar, J., Barrachina, S., García-Varea, I., Llorens, D., Martínez, C., Molau, S., Nevado, F., Pastor, M., Picó, D., Sanchis, A. and Tillmann, C. (2004), «Some approaches to statistical and finite-state speech-to-speech translation», *Computer Speech and Language*, vol. 18 (1), pp. 25–47. (Cited on page 32.)

Caseiro, D. A. (2003), *Finite-State Methods in Automatic Speech Recognition*, Ph.D. thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Portugal. (Cited on page 346.)

Castro, M. and Casacuberta, F. (1991), «The use of multilayer perceptrons in isolated word recognition», in Prieto, A., ed., «Artificial Neural Networks», vol. 540 of *Lecture Notes in Computer Science*, pp. 348–354, 10.1007/BFb0035913. (Cited on page 52.)

Castro, M., Polvoreda, V. and Prat, F. (2001a), «Connectionist N-gram Models by Using MLPs», in «Proceedings of the Second Workshop on Natural Language Processing and Neural Networks», pp. 16–22, Tokyo (Japan). (Cited on page 263.)

Castro, M. J. and Prat, F. (2003), «New Directions in Connectionist Language Modeling», in «Computational Methods in Neural Modeling», vol. 2686 of *Lecture Notes in Computer Science*, pp. 598–605, Springer-Verlag. (Cited on page 263.)

Castro, M. J., España, S., Marzal, A. and Salvador, I. (2001b), «Grapheme-to-phoneme conversion for the Spanish language», in «Pattern Recognition and Image Analysis. Proceedings of the IX Spanish Symposium on Pattern Recognition and Image Analysis», pp. 397–402, AERFAI. (Cited on page 641.)

Castro, M. J., España, S., Marzal, A. and Salvador, I. (2001c), «Transcriptor ortográfico-fonético para el castellano», *Procesamiento del Lenguaje Natural*, vol. 27, pp. 241–245, iSSN: 1135–5948. (Cited on page 641.)

Castro, M. J., Díaz, W., Ferri, F. J., Ruiz-Pinales, J., Jaime-Rivas, R., Blat, F., España, S., Aibar, P., Grau, S. and Griol, D. (2005), «A holistic classification system for check amounts based on neural networks with rejection», in «Proc. Int. Conf. on Pattern Recognition and Machine Intelligence», vol. 3776 of *Lecture Notes in Computer Science*, pp. 310–314. (Cited on pages 60, 606, and 639.)

Castro, M. J., España, S., Llorens, D., Marzal, A., Prat, F., Vilar, J. M. and Zamora, F. (2011), «Speech interaction in a multimodal tool for handwritten text transcription», in «ICMI», pp. 299–302. (Cited on pages 84, 381, and 641.)

Castro-Bleda, M., España, S., Prat, F., Vilar, J., Llorens, D., Marzal, A. and Zamora, F. (2014), «Human or Computer Assisted

Interactive Transcription: Automated Text Recognition, Text Annotation, and Scholarly Edition in the Twenty-First Century», *Mirabilia*, pp. 0247–253. (Cited on page 642.)

Castro-Bleda, M. J., España-Boquera, S. and Zamora-Martínez, F. (2009a), *Encyclopedia of Artificial Intelligence*, chap. Behaviour-based Clustering of Neural Networks, pp. 144–151. (Cited on page 639.)

Castro-Bleda, M. J., Díaz-Villanueva, W., Domínguez-Rubio, J. L. and España-Boquera, S. (2002), «Prediction and Discrimination of Pharmacological Activity by Using Committees of Artificial Neural Networks», in «Ensemble Methods for Learning Machines. Proc. of International School on Neural Nets "E. R. Caianiello"». (Cited on page 641.)

Castro-Bleda, M. J., España, S., Gorbe, J., Zamora, F., Llorens, D., Marzal, A., Prat, F. and Vilar, J. (2009b), «Improving a DTW-based recognition engine for on-line handwritten characters by using MLPs», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 1260–1264, IEEE. (Cited on page 638.)

Cesari, F., Franco, H., Myers, G. K. and Bratt, H. (2008), «MUESLI: multiple utterance error correction for a spoken language interface.», in «Conference of the International Speech Communication Association (Interspeech)», pp. 199–202. (Cited on page 80.)

Chan, A., Mosur, R., Rudnicky, A. and Sherwani, J. (2004), «Four-layer categorization scheme of fast GMM computation techniques in large vocabulary continuous speech recognition systems», in «International Conference on Spoken Language Processing (ICSLP)». (Cited on page 306.)

Chappelier, J.-C., Rajman, M., Aragüés, R. and Rozenknop, A. (1999), «Lattice parsing for speech recognition», in «Traitement Automatique du Langage Naturel», pp. 95–104, ATALA. (Cited on page 227.)

Charniak, E. (1993), *Statistical techniques for natural language parsing*, MIT Press. (Cited on page 29.)

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P. and Robinson, T. (2013), «One billion word benchmark for measuring progress in statistical language modeling», *arXiv preprint arXiv:1312.3005*. (Cited on pages 529 and 646.)

Chen, C., Gopinath, R., Monkowski, M., Picheny, M. and Shen, K. (1997), «New methods in continuous Mandarin speech recognition», in «European Conference on Speech Communication and Technology (Eurospeech)». (Cited on page 48.)

Chen, S., Beeferman, D. and Rosenfeld, R. (1998), «Evaluation metrics for language models», in «DARPA Broadcast News Transcription and Understanding Workshop». (Cited on page 248.)

Chen, S. F. and Goodman, J. (1998), «An Empirical Study of Smoothing Techniques for Language Modeling», Technical Report TR-10-98, Computer Science Group, Harvard University, revised March, 2004. (Cited on pages 243, 262, and 356.)

Chiang, D. (2004), *Evaluating grammar formalisms for applications to natural language processing and biological sequence analysis*, Ph.D. thesis, University of Pennsylvania. (Cited on page 194.)

Chiang, D. (2007), «Hierarchical phrase-based translation», *Computational Linguistics*, vol. 33 (2), pp. 201–228. (Cited on page 32.)

Chibelushi, C., Deravi, F. and Mason, J. (2002), «A review of speech-based bimodal recognition», *Multimedia, IEEE Transactions on*, vol. 4 (1), pp. 23–37. (Cited on page 41.)

Chien, L., Lee, L. and Chen, K. (1991), «An augmented chart data structure with efficient word lattice parsing scheme in speech recognition applications», *Speech Communication*, vol. 10 (2), pp. 129–144. (Cited on page 77.)

Chomsky, N. (1959), «On certain formal properties of grammars», *Information and control*, vol. 2 (2), pp. 137–167. (Cited on page 193.)

Chomsky, N. and Halle, M. (1968), *The sound pattern of English*, Harper and Row, New-York. (Cited on page 51.)

Chomsky, N. and Schützenberger, M. (1963), «The Algebraic Theory of Context-Free Languages», *Studies in Logic and the Foundations of Mathematics*, vol. 35, pp. 118–161. (Cited on pages 181 and 182.)

Chong, J., Gonina, E., You, K., Keutzer, K. and Parasians, L. (2010), «Exploring recognition network representations for efficient speech inference on highly parallel platforms», in «Conference of the International Speech Communication Association (Interspeech)». (Cited on page 412.)

Choueiter, G. F. (2009), *Linguistically-Motivated Sub-word Modeling with Applications to Speech Recognition*, Ph.D. thesis, Massachusetts Institute of Technology. (Cited on pages 250 and 270.)

Clarkson, P. and Robinson, T. (1999), «Towards improved language model evaluation measures», in «European Conference on Speech Communication and Technology (Eurospeech)», vol. 5, pp. 1927–1930. (Cited on page 248.)

Clarkson, P. and Rosenfeld, R. (1997), «Statistical Language Modeling using the CMU-Cambridge toolkit», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 2707–2711. (Cited on page 356.)

Cocke, J. and Schwartz, J. (1970), «Programming Languages and Their Compilers: Preliminary Notes», Tech. rep., Courant Institute of Mathematical Sciences, New York University. (Cited on page 227.)

Collobert, R., Kavukcuoglu, K. and Farabet, C. (2011), «Torch7: A Matlab-like Environment for Machine Learning», in «NIPS Workshop BigLearn». (Cited on page 324.)

Conway, M. E. (1963), «Design of a separable transition-diagram compiler», *Communications of the ACM*, vol. 6 (7), pp. 396–408. (Cited on pages 187 and 236.)

Coppersmith, D. and Winograd, S. (1987), «Matrix multiplication via arithmetic progressions», in «Proceedings of the nineteenth annual ACM symposium on Theory of computing», pp. 1–6, ACM. (Cited on page 231.)

Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009), *Introduction to algorithms (3rd ed.)*, The MIT press. (Cited on pages 233, 360, 405, 407, 408, 418, and 469.)

Cortes, C. and Mohri, M. (2000), «Context-Free Recognition with Weighted Automata», *Grammars*, vol. 3, pp. 133–150. (Cited on page 387.)

Côté, M., Lecolinet, E., Cheriet, M. and Suen, C. Y. (1998), «Automatic reading of cursive scripts using a reading model and perceptual concepts. The PERCEPTO system», *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 1, pp. 3–17. (Cited on page 555.)

Crego, J., Yvon, F. and Mariño, J. (2011), «Ncode: an Open Source Bilingual N-gram SMT Toolkit», *The Prague Bulletin of Mathematical Linguistics*, (96), pp. 49–58. (Cited on page 346.)

Crego, J. M., Mariño, J. B. and Gispert, A. D. (2004), «Finitestate-based and phrase-based statistical machine translation», in «Conference of the International Speech Communication Association (Interspeech)». (Cited on page 32.)

Dabney, J. B. and Harman, T. L. (1997), *Mastering SIMULINK*, Prentice Hall PTR. (Cited on page 325.)

Daciuk, J., Maurel, D. and Savary, A. (2005), «Dynamic Perfect Hashing with Finite-State Automata», in «Intelligent Information Processing and Web Mining», vol. 31 of *Advances in Soft Computing*, pp. 169–178. (Cited on page 344.)

Daciuk, J. and van Noord, G. (2004), «Finite automata for compact representation of tuple dictionaries», *Theoretical Computer Science*, vol. 313 (1), pp. 45–56. (Cited on page 357.)

Daines, D. (2011), *An Architecture for Scalable, Universal Speech Recognition*, Ph.D. thesis, Carnegie Mellon University. (Cited on pages 363, 411, and 412.)

Darwiche, A. (2009), *Modeling and reasoning with Bayesian networks*, Cambridge University Press. (Cited on pages 106, 121, 124, 125, 166, and 621.)

Davis, J. R. (1991), «Let your fingers do the spelling: Implicit disambiguation of words spelled with the telephone keypad», *Journal of The American Voice I/O Society*, vol. 9, pp. 9–57. (Cited on page 35.)

de Borda, J. (1781), *Mémoire sur les élections au scrutin in the Histoire de l'Académie Royale des Sciences*, Paris, France. (Cited on page 608.)

De Mori, R. (2007), «Spoken language understanding: a survey», in «Automatic Speech Recognition Understanding, 2007. ASRU. IEEE Workshop on», pp. 365–376. (Cited on pages 32 and 33.)

De Wachter, M., Matton, M., Demuynck, K., Wambacq, P., Cools, R. and Van Compernolle, D. (2007), «Template-based continuous speech recognition», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 15 (4), pp. 1377–1390. (Cited on page 163.)

Dechter, R. (1999), «Bucket Elimination: A Unifying Framework for Reasoning», *Artificial Intelligence*, vol. 113, p. 41–85. (Cited on page 123.)

Demuynck, K., Duchateau, J., Compernolle, D. V. and Wambacq, P. (2000), «An efficient search space representation for large vocabulary continuous speech recognition», *Speech Communication*, vol. 30 (1), pp. 37–53. (Cited on pages 413, 414, 415, 427, 448, and 453.)

Demuynck, K., Laureys, T., Compernolle, D. V. and hamme, H. V. (2003), «FLaVoR: a Flexible Architecture for LVCSR», in «Conference of the International Speech Communication Association (Interspeech)», pp. 1973–1976. (Cited on page 506.)

Deng, L. and Li, X. (2013), «Machine Learning Paradigms for Speech Recognition: An Overview», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 21 (5), pp. 1060–1089. (Cited on pages 18, 106, and 145.)

Deng, L., Yu, D. and Acero, A. (2006), «Structured speech modeling», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 14 (5), pp. 1492–1504. (Cited on pages 288, 289, 290, 300, 301, and 318.)

Deoras, A. (2011), *Search and decoding strategies for complex lexical modeling in LVCSR*, Ph.D. thesis, Johns Hopkins University. (Cited on page 279.)

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R. and Makhoul, J. (2014), «Fast and robust neural network joint models for statistical machine translation», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)». (Cited on pages 377 and 600.)

Dietz, L. (2010), «Directed factor graph notation for generative models», *Max Planck Institute for Informatics, Tech. Rep*. (Cited on page 117.)

Digalakis, V., Monaco, P. and Murveit, H. (1996), «Genones: generalized mixture tying in continuous hidden Markov model-based speech recognizers», *Speech and Audio Processing, IEEE Transactions on*, vol. 4 (4), pp. 281–289. (Cited on page 298.)

Digalakis, V., Neumeyer, L. and Perakakis, M. (1998), «Product-code vector quantization of cepstral parameters for speech recognition over the www», in «International Conference on Spoken Language Processing (ICSLP)». (Cited on page 305.)

Dijkstra, E. W. (1959), «A note on two problems in connexion with graphs», *Numerische mathematik*, vol. 1 (1), pp. 269–271. (Cited on pages 230, 233, and 407.)

Doetsch, P., Kozielski, M. and Ney, H. (2014), «Fast and Robust Training of Recurrent Neural Networks for Offline Handwriting Recognition», in «Frontiers in Handwriting Recognition (ICFHR), International Conference on», pp. 279–284. (Cited on page 296.)

Doetsch, P., Ney, H. *et al.* (2013), «Improvements in RWTH's System for Off-Line Handwriting Recognition», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 935–939, IEEE. (Cited on page 296.)

Doetsch, P., Ney, I. H. and Plahl, D. I. C. (2011), *Optimization of hidden markov models and neural networks*, Ph.D. thesis, Master's thesis, RWTH Aachen University, Aachen, Germany. (Cited on page 307.)

Domingos, P. (2012), «A few useful things to know about machine learning», *Communications of the ACM*, vol. 55 (10), pp. 78–87. (Cited on pages 106 and 114.)

Dreuw, P., Heigold, G. and Ney, H. (2011a), «Confidence- and margin-based MMI/MPE discriminative training for off-line handwriting recognition», *International Journal of Document Analysis and Recognition (IJDAR)*, pp. 1–16. (Cited on page 596.)

Dreuw, P., Doetsch, P., Plahl, C. and Ney, H. (2011b), «Hierarchical hybrid MLP/HMM or rather MLP features for a discriminatively trained Gaussian HMM: A comparison for offline handwriting recognition», in «Image Processing (ICIP), 2011 18th IEEE International Conference on», pp. 3541–3544. (Cited on page 596.)

Dreyer, M. and Marcu, D. (2012), «HyTER: meaning-equivalent semantics for translation evaluation», in «Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies», NAACL HLT '12, pp. 162–171. (Cited on page 237.)

Driscoll, J. R., Sarnak, N., Sleator, D. D. and Tarjan, R. E. (1989), «Making data structures persistent», *Journal of Computer and System Sciences*, vol. 38 (1), pp. 86–124. (Cited on page 390.)

Duda, R. O., Hart, P. E. and Stork, D. G. (2001), *Pattern Classification*, John Wiley and Sons, New York, NY, USA, second edn. (Cited on pages 106 and 114.)

Dunn, R. B., Reynolds, D. A. and Quatieri, T. F. (2000), «Approaches to Speaker Detection and Tracking in Conversational Speech», *Digital Signal Processing*, vol. 10 (1–3), pp. 93–112. (Cited on page 62.)

Dupont, P. (1993), «Dynamic Use Of Syntactical Knowledge In Continuous Speech Recognition», in «in Proceedings Third European Conference on Speech Communication and Technology», pp. 1959–1962. (Cited on pages 231, 232, and 245.)

Dupont, P. (1996), *Learning and Use of Language Models for Continuous Speech Recognition*, Ph.d. thesis, Ecole Nationale Supérieure des Télécommunications. (Cited on pages 231 and 232.)

Dupont, P. and Amengual, J.-C. (2000), «Smoothing Probabilistic Automata: An Error-Correcting Approach», in Oliveira, A., ed., «Grammatical Inference: Algorithms and Applications», vol. 1891 of *Lecture Notes in Computer Science*, pp. 51–64, Springer Berlin Heidelberg. (Cited on page 272.)

Dupont, P., Denis, F. and Esposito, Y. (2005), «Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms», *Pattern Recognition*, vol. 38, pp. 1349–1371. (Cited on page 265.)

Dupont, P. and Rosenfeld, R. (1997), «Lattice based language models», Tech. rep., DTIC Document. (Cited on pages 280, 355, and 622.)

Duvinage, M. and Parfait, J. (2010), «Reconnaissance vocale basée sur les phonèmes voisés», in «Actes des 28emes Journées d'Etude sur la Parole (JEP 2010)», pp. 257–260. (Cited on pages 50 and 423.)

Dyer, C. (2010), *A Formal Model of Ambiguity and its Applications in Machine Translation*, Ph.D. thesis, University of Maryland, College Park, Maryland, USA, advisor: Prof. Philip Resnik. (Cited on pages 169, 170, 174, 186, 194, 231, 236, and 237.)

Earley, J. (1970), «An efficient context-free parsing algorithm», *Communications of the ACM*, vol. 13 (2), pp. 94–102. (Cited on pages 191, 227, and 228.)

Egmont-Petersen, M., de Ridder, D. and Handels, H. (2002), «Image processing with neural networks – a review», *Pattern Recognition*, vol. 35 (10), pp. 2279–2301. (Cited on page 542.)

El-Yacoubi, A., Gilloux, M., Sabourin, R. and Suen, C. Y. (1999a), «An HMM-Based approach for off-line unconstrained handwritten word modeling and recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 21 (8), pp. 752–760. (Cited on page 579.)

El-Yacoubi, A., Gilloux, M., Sabourin, R. and Suen, C. Y. (1999b), «Unconstrained Handwritten Word Recognition using Hidden Markov Models», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 21 (8), pp. 752–760. (Cited on page 553.)

Eppstein, D. (1998), «Finding the k shortest paths», *SIAM J. Comput.*, vol. 28 (2), pp. 652–673. (Cited on page 76.)

Ésik, Z. and Kuich, W. (2007), *Modern Automata Theory*, Boston, MA. (Cited on pages 169, 176, 183, 184, and 185.)

España, S., Castro, M. J. and Hidalgo, J. L. (2004), «The SPARTACUS-Database: a Spanish Sentence Database for Offline Handwriting Recognition», in «Proceedings of the International Conference on Language Resources and Evaluation (LREC)», vol. I, pp. 227–230. (Cited on pages 68, 637, and 646.)

España-Boquera, S., Gorbe-Moya, J. and Zamora-Martínez, F. (2007), «Semiring Lattice Parsing Applied to CYK», in «Pattern Recognition and Image Analysis, IbPRIA», pp. 603–610, Springer. (Cited on pages 227, 230, and 637.)

Espana-Boquera, S., Zamora-Martínez, F., Castro-Bleda, M. and Gorbe-Moya, J. (2007), «Efficient BP algorithms for general feed-forward neural networks», in «Bio-inspired Modeling of Cognitive Tasks», pp. 327–336, Springer. (Cited on page 637.)

España-Boquera, S., Castro-Bleda, M. J., Zamora-Martínez, F. and Gorbe-Moya, J. (2007), «Efficient Viterbi algorithms for lexical tree based models», in «Advances in Nonlinear Speech Processing», pp. 179–187, Springer. (Cited on pages 413, 459, 474, 590, and 637.)

España-Boquera, S., Gorbe-Moya, J., Zamora-Martínez, F. and Castro-Bleda, M. J. (2010), «Hybrid HMM/ANN models for bimodal online and offline cursive word recognition», in «Recognizing Patterns in Signals, Speech, Images and Videos», pp. 14–21. (Cited on pages 572, 574, 578, 610, and 637.)

España-Boquera, S., Castro-Bleda, M., Gorbe-Moya, J. and Zamora-Martinez, F. (2011), «Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 33 (4), pp. 767–779. (Cited on pages 541, 543, 554, 557, 573, 578, 579, 582, 583, 584, 585, 587, 588, 589, 590, 591, 593, 596, 597, 600, 605, 609, 611, 612, 626, and 636.)

Espy-Wilson, C., Pruthi, T., Juneja, A. and Deshmukh, O. (2007), «Landmark-based Approach to Speech Recognition: An Alternative to HMMs», in «Conference of the International Speech Communication Association (Interspeech)», pp. 886–889. (Cited on page 51.)

Evermann, G. (1999), *Minimum Word Error Rate Decoding*, Master's thesis, Cambridge University. (Cited on pages 7 and 78.)

Evermann, G. and Woodland, P. (2000), «Posterior probability decoding, confidence estimation and system combination», in «Proc. Speech Transcription Workshop»vol. 27, Baltimore. (Cited on page 78.)

Federico, M., Bertoldi, N. and Cettolo, M. (2008), «IRSTLM: an open source toolkit for handling large scale language models.», in «Conference of the International Speech Communication Association (Interspeech)», pp. 1618–1621. (Cited on page 356.)

Federmann, C. (2007), *Very large language models for machine translation*, Master's thesis, Saarland University. Faculty of Natural Sciences and Technology I. Department of Computer Science. (Cited on page 344.)

Felzenszwalb, P. and McAllester, D. (2007), «The Generalized A* Architecture», *Journal of Artificial Intelligence Research*, vol. 29, pp. 153–190. (Cited on page 53.)

Fenton, J. (Saturday 29 July 2006), «Read my lips», *The Guardian*. (Cited on page 1.)

Ferretti, M., Maltese, G. and Scarci, S. (1989), «Language model and acoustic model information in probabilistic speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 707–710 vol.2. (Cited on page 249.)

Fink, G., Schillo, C., Kummert, F., Sagerer, G. *et al.* (1998), «Incremental speech recognition for multimodal interfaces», in «Proceedings 24th Annual Conference of the IEEE Industrial Electronics Society», vol. 4, pp. 2012–2017. (Cited on page 325.)

Finke, M. and Waibel, A. (1997), «Speaking Mode Dependent Pronunciation Modeling In Large Vocabulary Conversational Speech Recognition», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 2379–2382. (Cited on page 252.)

Finke, M., Fritsch, J., Koll, D. and Waibel, A. (1999), «Modeling And Efficient Decoding Of Large Vocabulary Conversational Speech», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 467–470. (Cited on pages 272, 346, 423, 427, and 516.)

Finkel, J. R., Manning, C. D. and Ng, A. Y. (2006), «Solving the problem of cascading errors: approximate Bayesian inference for linguistic annotation pipelines», in «Proc. of Empirical Methods in Natural Language Processing (EMNLP)», pp. 618–626. (Cited on page 53.)

Fischer, A. (2012), *Handwriting Recognition in Historical Documents*, Ph.D. thesis, Technische Universität München. Fakultät für Informatik. (Cited on pages xxiii, 52, 77, 148, 157, 159, and 309.)

Fischer, A., Wüthrich, M., Liwicki, M., Frinken, V., Bunke, H., Viehhauser, G. and Stolz, M. (2009), «Automatic transcription of handwritten medieval documents», in «Proceedings of the 2009 15th International Conference on Virtual Systems and Multimedia», pp. 137–142, IEEE. (Cited on page 628.)

Fischer, A., Frinken, V., Fornés, A. and Bunke, H. (2011), «Transcription alignment of Latin manuscripts using hidden Markov models», in «Proceedings of the 2011 Workshop on Historical Document Imaging and Processing», pp. 29–36, ACM. (Cited on page 628.)

Fischer, A., Keller, A., Frinken, V. and Bunke, H. (2012), «Lexicon-free handwritten word spotting using character {HMMs}», *Pattern Recognition Letters*, vol. 33 (7), pp. 934–942. (Cited on page 71.)

Fiscus, J. (1997), «A post-processing system to yield reduced word error rates: Recognizer Output Voting Error Reduction (ROVER)», in «Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on», pp. 347–354. (Cited on pages 81, 560, and 594.)

Fissore, L., Laface, P., Micca, G. and Ravera, F. (1996), «Vocabulary Independent Acoustic-Phonetic Modeling for Continuous Speech Recognition», *Proc. of European Signal Processing Conference (EUSIPCO)*. (Cited on page 45.)

Forney, J., G.D. (2001), «Codes on graphs: normal realizations», *Information Theory, IEEE Transactions on*, vol. 47 (2), pp. 520–548. (Cited on page 117.)

Fosler-Lussier, E. and Morgan, N. (1999), «Effects of speaking rate and word frequency on pronunciations in convertional speech», *Speech Communication*, vol. 29 (2-4), pp. 137–158. (Cited on pages 54 and 259.)

Fousek, P. and Hermansky, H. (2006), «Towards ASR Based on Hierarchical Posterior-Based Keyword Recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 433–436. (Cited on page 72.)

Francis, W. and Kucera, H. (1979), «Brown Corpus Manual, Manual of Information to accompany A Standard Corpus of Present-Day Edited American English», Tech. rep., Department of Linguistics, Brown University, Providence, Rhode Island, US. (Cited on page 581.)

Frankel, J. (2003), *Linear dynamic models for automatic speech recognition*, Ph.D. thesis, The University of Edinburgh. College of Science and Engineering. School of Informatics. (Cited on pages 300 and 303.)

François, A. and Medioni, G. (2001), «A Modular Software Architecture for Real-Time Video Processing», in Schiele, B. and Sagerer, G., eds., «Computer Vision Systems», vol. 2095 of *Lecture Notes in Computer Science*, pp. 35–49. (Cited on page 325.)

Frey, B. J. (2002), «Extending factor graphs so as to unify directed and undirected graphical models», in «Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence», pp. 257–264, Morgan Kaufmann Publishers Inc. (Cited on pages 117 and 118.)

Frinken, V., Zamora-Martinez, F., Espana-Boquera, S., Castro-Bleda, M. J., Fischer, A. and Bunke, H. (2012a), «Long-short term memory neural networks language modeling for handwriting recognition», in «International Conference on Pattern Recognition (ICPR)», pp. 701–704, IEEE. (Cited on page 640.)

Frinken, V., Fischer, A., Manmatha, R. and Bunke, H. (2012b), «A Novel Word Spotting Method Based on Recurrent Neural Networks», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 34 (2), pp. 211–224. (Cited on page 158.)

Gales, M. and Flego, F. (2010), «Discriminative classifiers with adaptive kernels for noise robust speech recognition», *Computer Speech and Language*, vol. 24 (4), pp. 648–662. (Cited on page 164.)

Gales, M., Knill, K. and Young, S. (1999), «State-based Gaussian selection in large vocabulary continuous speech recognition using HMMs», *Speech and Audio Processing, IEEE Transactions on*, vol. 7 (2), pp. 152–161. (Cited on page 306.)

Gall, F. L. (2014), «Powers of Tensors and Fast Matrix Multiplication», *Computing Research Repository (CoRR)*, vol. abs/1401.7714. (Cited on page 231.)

Gallo, G., Longo, G. and Pallottino, S. (1993), «Directed Hypergraphs and Applications», *Discrete Applied Mathematics*, vol. 42 (2), pp. 177–201. (Cited on page 230.)

Ganapathiraju, A., Hamaker, J., Picone, J., Ordowski, M. and Doddington, G. (2001), «Syllable-based large vocabulary continuous speech recognition», *Speech and Audio Processing, IEEE Transactions on*, vol. 9 (4), pp. 358–366. (Cited on page 45.)

García, P. and Vidal, E. (1990), «Inference of k-testable languages in the strict sense and application to syntactic pattern recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 12 (9), pp. 920–925. (Cited on pages 244, 261, and 345.)

García, P., Segarra, E., Vidal, E. and Galiano, I. (1990), «On the use of the morphic generator grammatical inference (MGGI) methodology in automatic speech recognition», *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 4 (04), pp. 667–685. (Cited on page 269.)

Gemello, R., Albesano, D. and Mana, F. (1997), «Continuous speech recognition with neural networks and stationary-transitional acoustic units», in «Neural Networks,1997., International Conference on», vol. 4, pp. 2107–2111 vol.4. (Cited on pages 45, 313, and 314.)

Genzel, D. (2005), «A Paragraph Boundary Detection System», in Gelbukh, A., ed., «Computational Linguistics and Intelligent Text Processing», vol. 3406 of *Lecture Notes in Computer Science*, pp. 816–826. (Cited on page 28.)

Georgila, K., Sgarbas, K. N., Fakotakis, N. and Kokkinakis, G. (2000), «Fast very large vocabulary recognition based on compact DAWG-structured language models.», in «Conference of the International Speech Communication Association (Interspeech)», pp. 987–990. (Cited on page 415.)

Ghahramani, Z. and Jordan, M. I. (1997), «Factorial Hidden Markov Models», *Machine Learning*, vol. 29, pp. 245–273. (Cited on page 256.)

Gibrat, J.-P. and Dehaene, S. (2012), «Les Chemins de la Lecture», Documentary film. (Cited on pages 1 and 4.)

Gillick, D., Wegmann, S. and Gillick, L. (2012), «Discriminative training for speech recognition is compensating for statistical dependence in the HMM framework», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 4745–4748. (Cited on page 290.)

Gillick, L. and Cox, S. (1989), «Some statistical issues in the comparison of speech recognition algorithms», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 532–535 vol.1. (Cited on page 97.)

Gilloux, M., Lemarié, B. and Leroux, M. (1995), «A hybrid radial basis function network/hidden Markov model handwritten word recognition system», in «International Conference on Document Analysis and Recognition (ICDAR)», vol. 1, pp. 394–397, IEEE. (Cited on page 307.)

Giménez, A. and Juan, A. (2009), «Bernoulli HMMs at Subword Level for Handwritten Word Recognition», in «Proceedings of the 4th Iberian Conference on Pattern Recognition and Image Analysis», IbPRIA '09, pp. 497–504. (Cited on page 305.)

Ginsburg, S. and Rice, H. G. (1962), «Two Families of Languages Related to ALGOL», *J. ACM*, vol. 9, pp. 350–371. (Cited on pages 175, 182, and 236.)

Glass, J., Chang, J. and McCandless, M. (1996), «A probabilistic framework for feature-based speech recognition», in «International Conference on Spoken Language Processing (ICSLP)», vol. 4, pp. 2277–2280, IEEE. (Cited on page 151.)

Glass, J. R. (2003), «A probabilistic framework for segment-based speech recognition», *Computer Speech and Language*, vol. 17, pp. 137–152. (Cited on pages 51, 150, 151, 152, 153, and 286.)

Godin, C. and Lockwood, P. (1989), «DTW schemes for continuous speech recognition: a unified view», *Computer Speech and Language*, vol. 3 (2), pp. 169–198. (Cited on pages 7 and 165.)

Goel, V., Mishra, A., Alahari, K. and Jawahar, C. (2013), «Whole is greater than sum of parts: Recognizing scene text words», in «International Conference on Document Analysis and Recognition (ICDAR)». (Cited on page 164.)

Gómez-Rodríguez, C. (2014), «Finding the smallest binarization of a CFG is NP-hard», *Journal of Computer and System Sciences*, vol. 80 (4), pp. 796–805. (Cited on page 193.)

Goodman, J. (1998), *Parsing Inside-Out*, Ph.D. thesis. (Cited on page 173.)

Goodman, J. (1999), «Semiring parsing», *Computational Linguistics*, vol. 25 (4), pp. 573–605. (Cited on pages 7, 169, 172, 194, 227, 230, and 236.)

Goodman, J. (2001), «A bit of progress in language modeling», *Computer Speech and Language*, vol. 15 (4), pp. 403–434. (Cited on pages 263, 267, 278, and 371.)

Gorbe-Moya, J. (2005), *Preproceso, parametrización y reconocimiento de escritura manuscrita mediante técnicas geométricas y estadísticas*, Master's thesis, Facultad de Informàtica. Universidad Politécnica de Valencia, supervised by Salvador España. (Cited on pages 550, 559, and 639.)

Gorbe-Moya, J., España-Boquera, S., Zamora-Martínez, F. and Castro-Bleda, M. J. (2008), «Handwritten Text Normalization by using Local Extrema Classification», in «International Workshop on Pattern Recognition in Information Systems», pp. 164–172. (Cited on pages 573, 580, and 637.)

Graham, S. L., Harrison, M. A. and Ruzzo, W. L. (1980), «An Improved Context-Free Recognizer», *ACM Transactions on Programming Languages and Systems*, vol. 2 (3), pp. 415–462. (Cited on pages 191 and 229.)

Graves, A. (2008), *Supervised Sequence Labelling with Recurrent Neural Networks*, Ph.D. thesis, Technische Universität München. Fakultät für Informatik. (Cited on pages 6 and 58.)

Graves, A., Fernández, S. and Schmidhuber, J. (2007), «Multi-Dimensional Recurrent Neural Networks», in «Proceedings of the International Conference on Artificial Neural Networks», ICANN'07, pp. 549–558. (Cited on page 566.)

Graves, A. and Schmidhuber, J. (2005), «Framewise phoneme classification with bidirectional LSTM and other neural network architectures», *Neural Networks*, vol. 18 (5-6), pp. 602–610. (Cited on page 158.)

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H. and Schmidhuber, J. (2009), «A Novel Connectionist System for Unconstrained Handwriting Recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 31 (5), pp. 855–868. (Cited on pages 159, 580, 589, 591, 593, 596, 597, 602, and 647.)

Grézl, F., Karafiát, M., Kontár, S. and Cernocky, J. (2007), «Probabilistic and bottle-neck features for LVCSR of meetings», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 4, pp. IV–757. (Cited on pages 308 and 311.)

Grosicki, E. and El Abed, H. (2009), «Icdar 2009 handwriting recognition competition», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 1398–1402, IEEE. (Cited on pages 575, 604, and 628.)

Grune, D. and Jacobs, C. (2008), *Parsing techniques: a practical guide*, Springer-Verlag New York Inc. (Cited on page 167.)

Grune, D. and Jacobs, C. J. (2007), *Parsing Techniques*, Monographs in Computer Science. Springer, 2nd ed. (Cited on pages 225 and 228.)

Guillevic, D. and Suen, C. Y. (1994), «Cursive Script Recognition: A Sentence Level Recognition Scheme», in «In International Workshop on Frontiers of Handwriting Recognition», pp. 216–223. (Cited on page 555.)

Gutkin, A. (2000), *Log-Linear Interpolation of Language Models*, MPhil. thesis, Department of Engineering, University of Cambridge, UK. (Cited on page 262.)

Gutmann, M. and Hyvärinen, A. (2010), «Noise-contrastive estimation: A new estimation principle for unnormalized statistical models», in «International Conference on Artificial Intelligence and Statistics», pp. 297–304. (Cited on page 370.)

Guyon, I., Schomaker, L., Plamondon, R., Liberman, M. and Janet, S. (1994), «UNIPEN project of on-line data exchange and recognizer benchmarks», in «Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on», vol. 2, pp. 29–33, IEEE. (Cited on page 646.)

Hahn, S., Dinarelli, M., Raymond, C., Lefevre, F., Lehnen, P., De Mori, R., Moschitti, A., Ney, H. and Riccardi, G. (2010), «Comparing Stochastic Approaches to Spoken Language Understanding in Multiple Languages», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. PP (99), p. 1. (Cited on page 33.)

Hakkani-Tur, D. and Riccardi, G. (2003), «A general algorithm for word graph matrix decomposition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 596–599 vol.1. (Cited on page 78.)

Han, Y. S. and Choi, K.-S. (1994), «A reestimation algorithm for Probabilistic Recursive Transition Network», in «Proceedings of the 15th conference on Computational linguistics - Volume 2», COLING '94, pp. 859–864. (Cited on page 187.)

Hanazawa, K., Minami, Y. and Furui, S. (1997), «An efficient search method for large-vocabulary continuous-speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 3, pp. 1787–1790. (Cited on page 415.)

Hanneforth, T. (2011), «A Practical Algorithm for Intersecting Weighted Context-free Grammars with Finite-State Automata», in «International Workshop Finite State Methods and Natural Language Processing», p. 57. (Cited on pages 231 and 236.)

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A. *et al.* (2014), «DeepSpeech: Scaling up end-to-end speech recognition», *arXiv preprint arXiv:1412.5567*. (Cited on page 311.)

Harte, N., Vaseghi, S. and Mccourt, P. (1998), «A novel model for phoneme recognition using phonetically derived features», in «Eusipco: European signal processing conference», pp. 1489–1492. (Cited on page 52.)

Hauenstein, A. (1993), «Architecture of a 10,000 Word Real Time Speech Recognizer», in «Third European Conference on Speech Communication and Technology». (Cited on page 413.)

Heafield, K. (2011), «KenLM: Faster and smaller language model queries», in «Proc. of the Sixth Workshop on Statistical Machine Translation». (Cited on pages 356, 357, 361, 363, 526, and 529.)

Hemerik, K. (2009), «Towards a Taxonomy for ECFG and RRPG Parsing», in «Proceedings of the 3rd International Conference on Language and Automata Theory and Applications», LATA '09, pp. 410–421. (Cited on page 188.)

Hennig, A. and Sherkat, N. (2002), «Exploiting zoning based on approximating splines in cursive script recognition», *Pattern Recognition*, vol. 35 (2), pp. 445–454. (Cited on page 546.)

Hermansky, H. (2003), «TRAP-TANDEM: Data-driven extraction of temporal features from speech», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 255–260. (Cited on page 308.)

Hermansky, H., Ellis, D. and Sharma, S. (2000), «Tandem connectionist feature extraction for conventional HMM systems», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 3, pp. 1635 –1638 vol.3. (Cited on pages 308 and 311.)

Hetherington, L. (2001), «An Eficient Implementation of Phonological Rules using Finite-State Transducers», in «Conference of the International Speech Communication Association (Interspeech)», pp. 1599–1602. (Cited on page 48.)

Hidalgo, J. L., España, S., Castro, M. J. and Pérez, J. A. (2005), «Enhancement and cleaning of handwritten data by using neural networks», in «Pattern Recognition and Image Analysis», vol. 3522 of *Lecture Notes in Computer Science*, pp. 376–383. (Cited on pages 543, 626, and 637.)

Hinton, G. E., Vinyals, O. and Dean, J. (2014), «Distilling the knowledge in a neural network», in «NIPS 2014 Deep Learning Workshop». (Cited on pages 258, 370, 373, and 379.)

Hiroaki, S. (1982), «A Generalized Two-Level DP-Matching Algorithm for Continuous Speech Recognition», *IEICE Transactions*, vol. E65-E (11), pp. 649–656. (Cited on pages 142 and 163.)

Hirsimäki, T., Creutz, M., Siivola, V., Kurimo, M., Virpioja, S. and Pylkkönen, J. (2006), «Unlimited vocabulary speech recognition with morph language models applied to Finnish», *Computer Speech and Language*, vol. 20 (4), pp. 515–541. (Cited on pages 48, 251, and 293.)

Hoffmeister, B., Klein, T., Schlüter, R. and Ney, H. (2006), «Frame based system combination and a comparison with weighted ROVER and CNC.», in «Conference of the International Speech Communication Association (Interspeech)». (Cited on page 78.)

Holmes, W. J. and Russell, M. J. (1999), «Probabilistic-trajectory segmental HMMs», *Computer Speech and Language*, vol. 13 (1), pp. 3–37. (Cited on page 291.)

Hopcroft, J. and Ullman, J. (1979), «Introduction to automata theory, languages and computation», *Addison-Wesley series in computer science*. (Cited on pages 179, 186, 194, and 200.)

Hori, T., Hori, C., Minami, Y. and Nakamura, A. (2007), «Efficient WFST-Based One-Pass Decoding With On-The-Fly Hypothesis Rescoring in Extremely Large Vocabulary Continuous Speech Recognition», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 15 (4), pp. 1352 –1365. (Cited on page 346.)

Hsu, B.-J. and Glass, J. (2008), «Iterative language model estimation: efficient data structure & algorithms», in «Conference of the International Speech Communication Association (Interspeech)», vol. 8, pp. 1–4. (Cited on page 356.)

Hu, H. (2012), *Towards an improved model of dynamics for speech recognition and synthesis*, Ph.D. thesis, University of Birmingham. (Cited on pages 300 and 302.)

Huang, B., Zhang, Y. and Kechadi, M. (2009), «Preprocessing Techniques for Online Handwriting Recognition», in Nedjah, N., de Macedo Mourelle, L., Kacprzyk, J., França, F. and de De Souza, A., eds., «Intelligent Text Categorization and Clustering», vol. 164 of *Studies in Computational Intelligence*, pp. 25–45. (Cited on page 570.)

Huang, C. and Srihari, S. N. (2008), «Word segmentation of off-line handwritten documents», in «Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series», vol. 6815, p. 13. (Cited on page 567.)

Huang, L. (2008a), «Advanced dynamic programming in semiring and hypergraph frameworks», *COLING, Manchester, UK*. (Cited on pages 230 and 235.)

Huang, L. (2008b), *Forest-based algorithms in natural language processing*, Ph.D. thesis, University of Pennsylvania. (Cited on pages 169 and 237.)

Huang, X., Acero, A., Alleva, F., Hwang, M.-Y., Jiang, L. and Mahajan, M. (1995), «Microsoft Windows highly intelligent speech recognizer: Whisper», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 93–96. (Cited on page 424.)

Huang, X. D. and Jack, M. A. (1989), «Semi-continuous hidden Markov models for speech signals», *Computer Speech and Language*, vol. 3 (3), pp. 239–251. (Cited on page 306.)

Huang, Z., Weng, C., Li, K., Cheng, Y.-C. and Lee, C.-H. (2014), «Deep learning vector quantization for acoustic information retrieval», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 1350–1354. (Cited on page 311.)

Huijbregts, M. A. H. (2008), *Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled*, Ph.D. thesis, Univ. of Twente. (Cited on pages 344, 357, 416, 427, 430, 431, 453, and 516.)

Hulden, M. (2011), «Parsing CFGs and PCFGs with a Chomsky-Schützenberger representation», in «Human Language Technology. Challenges for Computer Science and Linguistics», pp. 151–160. (Cited on pages 181 and 231.)

Hull, J. J. (1994), «A database for handwritten text recognition research», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 16 (5), pp. 550–554. (Cited on page 646.)

Hunt, M. J. (1990), «Figures of merit for assessing connected-word recognisers», *Speech Communication*, vol. 9 (4), pp. 329–336. (Cited on page 97.)

Hwang, M. and Huang, X. (1992), «Subphonetic modeling with Markov states-Senone», vol. 1, pp. 33 –36 vol.1. (Cited on pages 45 and 298.)

Hwang, M.-Y., Huang, X. and Alleva, F. A. (1996), «Predicting unseen triphones with senones», vol. 4, pp. 412–419. (Cited on page 313.)

Ierusalimschy, R., de Figueiredo, L. H. and Filho, W. C. (1996), «Lua — an extensible extension language», *Softw. Pract. Exper.*, vol. 26, pp. 635–652. (Cited on page 324.)

Iglesias, G., Allauzen, C., Byrne, W., de Gispert, A. and Riley, M. (2011), «Hierarchical Phrase-based Translation Representations», in «Proc. of Empirical Methods in Natural Language Processing (EMNLP)», pp. 1373–1383. (Cited on page 237.)

Imai, T., Kobayashi, A., Sato, S., Tanaka, H. and Ando, A. (2000), «Progressive 2-pass decoder for real-time broadcast news captioning», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 3, pp. 1559–1562. (Cited on page 9.)

Iyer, R., Ostendorf, M. and Meteer, M. (1997), «Analyzing and predicting language model improvements», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 254–261. (Cited on page 248.)

Jaakkola, T. and Haussler, D. (1998), «Exploiting Generative Models in Discriminative Classifiers», in «Advances in Neural Information Processing Systems (NIPS)», pp. 487–493. (Cited on page 52.)

Jaeger, S., Manke, S. and Waibel, A. (2000), «Npen++: An On-Line Handwriting Recognition System», in «International Workshop on Frontiers in Handwriting Recognition (IWFHR)», pp. 249–260. (Cited on page 570.)

Jane, C. (1998), *Near-miss modeling: A segment-based approach to speech recognition*, Ph.D. thesis, Massachusetts Institute of Technology. (Cited on page 153.)

Jansen, A. and Niyogi, P. (2009), «Point process models for event-based speech recognition», *Speech Communication*, vol. 51 (12), pp. 1155–1168. (Cited on page 45.)

Jelinek, B., Zheng, F., Parihar, N., Hamaker, J. and Picone, J. (2001), «Generalized hierarchical search in the ISIP ASR system», in «Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on», vol. 2, pp. 1553–1556. (Cited on page 416.)

Jelinek, F. (1976), «Continuous speech recognition by statistical methods», *Proceedings of the IEEE*, vol. 64 (4), pp. 532–556. (Cited on page 286.)

Jelinek, F. (1998), *Statistical Methods for Speech Recognition*, The MIT Press. (Cited on pages 292, 405, and 408.)

Jelinek, F. and Chelba, C. (2000), «Structured Language Modeling for Speech Recognition», *Computer Speech and Language*, vol. 14 (4), pp. 283–332. (Cited on page 267.)

Jelinek, F. and Lafferty, J. D. (1991), «Computation of the probability of initial substring generation by stochastic context-free grammars», *Computational Linguistics*, vol. 17 (3), pp. 315–323. (Cited on page 243.)

Jelinek, F., Mercer, R. and Roukos, S. (1990), «Classifying words for improved statistical language models», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 621–624. (Cited on page 256.)

Jelinek, F., Mercer, R., Bahl, L. and Baker, J. (1977), «Perplexity—a measure of the difficulty of speech recognition tasks», *The Journal of the Acoustical Society of America*, vol. 62 (S1), pp. S63–S63. (Cited on page 246.)

Jelinek, F., Merialdo, B., Roukos, S. and Strauss, M. (1991), «A Dynamic Language Model for Speech Recognition.», *HLT*, vol. 91, pp. 293–295. (Cited on page 260.)

Jeng, F.-C. and Woods, J. W. (1987), «On the relationship of the Markov mesh to the NSHP Markov chain», *Pattern Recognition Letters*, vol. 5 (4), pp. 273–279. (Cited on page 309.)

Jensen, K., Wirth, N., Mickel, A. B. and Miner, J. F. (1975), *Pascal: user manual and report*, vol. 3, springer-Verlag New York. (Cited on page 188.)

Jeong, M. and Lee, G. G. (2008), «Practical use of non-local features for statistical spoken language understanding», *Computer Speech and Language*, vol. 22 (2), pp. 148–170. (Cited on page 32.)

Jim, T. and Mandelbaum, Y. (2010), «Efficient earley parsing with regular right-hand sides», *Electronic Notes in Theoretical Computer Science*, vol. 253 (7), pp. 135–148. (Cited on page 229.)

Jimenez, V. and Marzal, A. (2003), «A Lazy Version of Eppstein's K Shortest Paths Algorithm», in «Experimental and efficient algorithms: second international workshop, WEA 2003, Ascona, Switzerland, May 26-28, 2003: proceedings», vol. 2, p. 179, Springer Verlag. (Cited on page 511.)

Johansson, S., Atwell, E., Garside, R. and Leech, G. (1986), «The Tagged LOB Corpus: User's Manual», Tech. rep., Norwegian Computing Centre for the Humanities, Bergen, Norway. (Cited on pages 526, 580, 581, 645, and 647.)

Johnson, G. W. (1997), *LabVIEW Graphical Programming: Practical Applications in Instrumentation and Control*, McGraw-Hill School Education Group, 2nd edn. (Cited on page 325.)

Johnson, M. and Harper, M. (1999a), «Near minimal weighted word graphs for post-processing speech», . (Cited on page 493.)

Johnson, M. T. and Harper, M. P. (1999b), «Near Minimal Weighted Word Graphs For Post-Processing Speech», in «In 1999 Int. Workshop on Automatic Speech Recognition and Understanding». (Cited on page 77.)

Jolliffe, I. and MyiLibrary (2002), *Principal component analysis*, vol. 2, Wiley Online Library. (Cited on page 52.)

Juang, B., Rabiner, L., Levinson, S. and Sondhi, M. (1985), «Recent developments in the application of hidden Markov models to speaker-independent isolated word recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 10, pp. 9–12. (Cited on page 295.)

Jurafsky, D., Wooters, C., Segal, J., Stolcke, A., Fosler, E., Tajchaman, G. and Morgan, N. (1995), «Using a stochastic context-free grammar as a language model for speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 189–192. (Cited on pages 232 and 245.)

Jurafsky, D., Martin, J., Kehler, A., Vander Linden, K. and Ward, N. (2000), *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, vol. 163, MIT Press. (Cited on page 42.)

Justo-Blanco, R. (2009), *Modelos de lenguaje jerárquicos basados en clases de phrases: formulación, aprendizaje y decodificación*, Ph.D. thesis, UPV/EHU. (Cited on page 252.)

Kahn, G. (1974), «The semantics of a simple language for parallel programming», in Rosenfeld, J. L., ed., «Proceedings of IFIP Congress 74», pp. 471–475. (Cited on pages 324 and 325.)

Kaltenmeier, A., Caesar, T., Gloger, J. and Mandler, E. (1993), «Sophisticated topology of hidden Markov models for cursive script recognition», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 139–142. (Cited on pages 2 and 566.)

Kanthak, S. and Ney, H. (2004), «FSA: An Efficient and Flexible C++ Toolkit for Finite State Automata Using On-Demand Computation», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», pp. 510–517. (Cited on page 346.)

Kaplan, R. M. and Bresnan, J. (1982), «Lexical-functional grammar: A formal system for grammatical representation», *Formal Issues in Lexical-Functional Grammar*, pp. 29–130. (Cited on page 188.)

Kaplan, R. M. and Kay, M. (1994), «Regular models of phonological rule systems», *Comput. Linguist.*, vol. 20 (3), pp. 331–378. (Cited on page 48.)

Karat, C.-M., Halverson, C., Horn, D. and Karat, J. (1999), «Patterns of entry and correction in large vocabulary continuous speech recognition systems», in «Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit», CHI '99, pp. 568–575. (Cited on page 34.)

Kasami, T. (1965), «An efficient recognition and syntax analysis algorithm for context-free languages», Tech. rep., Air Force Cambridge Research Laboratory. (Cited on page 227.)

Kask, K., Dechter, R., Larrosa, J. and Dechter, A. (2005), «Unifying tree decompositions for reasoning in graphical models», in «Artificial Intelligence», vol. 166, pp. 165–193. (Cited on pages 121, 124, and 125.)

Katagiri, S. and Lee, C.-H. (1993), «A new hybrid algorithm for speech recognition based on HMM segmentation and learning vector quantization», *Speech and Audio Processing, IEEE Transactions on*, vol. 1 (4), pp. 421–430. (Cited on page 308.)

Kato, T., Fujita, K. and Nishizawa, N. (2010), «An efficient beam pruning with a reward considering the potential to reach various words on a lexical tree», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 4930–4933. (Cited on pages 416 and 433.)

Katz, S. (1987), «Estimation of Probabilities From Sparse Data for the Language Model Component of a Speech Recognizer», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 34 (3), pp. 400–401. (Cited on page 262.)

Kavallieratou, E., Dromazou, N., Fakotakis, N. and Kokkinakis, G. (2003), «An integrated system for handwritten document image processing», *International journal of pattern recognition and artificial intelligence*, vol. 17 (4), pp. 617–636. (Cited on page 552.)

Kawahara, T., Lee, A., Takeda, K., Itou, K. and Shikano, K. (2004), «Recent progress of open-source LVCSR engine julius and Japanese model repository», in «Conference of the International Speech Communication Association (Interspeech)», pp. 3069–3072. (Cited on page 77.)

Kay, M. (1980), «Algorithm schemata and data structures in syntactic processing», *Technical Report CSL80-12*. (Cited on page 227.)

Kennard, D. J. and Barrett, W. A. (2006), «Separating Lines of Text in Free-Form Handwritten Historical Documents», in «Proceedings of the Second International Conference on Document Image Analysis for Libraries», pp. 12–23. (Cited on page 552.)

Kenny, P., Hollan, R., Gupta, V., Lennig, M., Mermelstein, P. and O'Shaughnessy, D. (1991), «A*-admissible heuristics for rapid lexical access», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 689–692. (Cited on page 422.)

Khalilov, M., Fonollosa, J. A. R., Zamora-Martínez, F., Castro-Bleda, M. J. and España-Boquera, S. (2008a), «Arabic-English translation improvement by target-side neural network language modeling», in (ELRA), E. L. R. A., ed., «Proceedings of the International Conference on Language Resources and Evaluation (LREC)». (Cited on page 640.)

Khalilov, M., Fonollosa, J., Zamora-Martinez, F., Castro-Bleda, M. and España-Boquera, S. (2008b), «Neural Network Language Models for Translation with Limited Data», in «Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on», vol. 2, pp. 445–451. (Cited on page 640.)

Khalilov, M., Fonollosa, J. A., Castro Bleda, M. J., España Boquera, S. *et al.* (2013), «Neural network language models to select the best translation», *Computational Linguistics in the Netherlands Journal*, (3), pp. 217–233. (Cited on page 641.)

Kim, N. S. and Un, C. (1997), «Frame-correlated hidden Markov model based on extended logarithmic pool», *Speech and Audio Processing, IEEE Transactions on*, vol. 5 (2), pp. 149–160. (Cited on page 290.)

Kingsbury, B. (2009), «Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling», *Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)*, vol. 0, pp. 3761–3764. (Cited on pages 613 and 635.)

Kisun, Y., Jike, C., Youngmin, Y., Ekaterina, G., Christopher, H., Wonyong, S. and Kurt, K. (2009), «Parallel Scalability in Speech Recognition: Inference Engine in Large Vocabulary Continuous Speech Recogniton», *Signal Processing Magazine, IEEE*, vol. 26, pp. 124–135. (Cited on page 405.)

Kita, K., Kawabata, T. and Saito, H. (1989), «HMM Continuous Speech Recognition Using Predictive LR Parsing», in «In IEEE International Conference on Acoustics, Speech and Signal Processing», pp. 703–706. (Cited on page 231.)

Kitaoka, N., Takahashi, N. and Nakagawa, S. (2005), «Large-vocabulary continuous speech recognition using linear lexicon search and 1-best approximation tree-structured lexicon searching», *Syst. Comput. Japan*, vol. 36, pp. 31–39. (Cited on pages 412 and 428.)

Klakow, D. and Peters, J. (2002), «Testing the correlation of word error rate and perplexity», *Speech Communication*, vol. 38 (1), pp. 19–28. (Cited on page 248.)

Klein, D. and Manning, C. D. (2001a), «An $O(n^3)$ Agenda-Based Chart Parser for Arbitrary Probabilistic Context-Free Grammars», Technical Report 2001-16, Stanford InfoLab. (Cited on page 227.)

Klein, D. and Manning, C. D. (2001b), «Parsing and Hypergraphs», in «The Seventh Internation Workshop on Parsing Technologies». (Cited on pages 169 and 230.)

Klovstad, J. and Mondshein, L. (1975), «The CASPERS linguistic analysis system», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 23 (1), pp. 118–123. (Cited on page 412.)

Kneser, R. and Ney, H. (1995), «Improved backing-off for m-gram language modeling», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 181–184. (Cited on pages 262 and 263.)

Knuth, D. (1977), «A generalization of Dijkstra's algorithm», *Inf. Process. Lett.*, vol. 6, pp. 1–5. (Cited on pages 230 and 237.)

Knuth, D. E. (1965), «On the translation of languages from left to right», *Information and control*, vol. 8 (6), pp. 607–639. (Cited on page 228.)

Knuth, D. E. (1998), *The art of computer programming: sorting and searching*, vol. 3, Pearson Education. (Cited on page 468.)

Koehn, P. (2004), «Statistical significance tests for machine translation evaluation», in «Proc. of Empirical Methods in Natural Language Processing (EMNLP)», pp. 388–395. (Cited on page 534.)

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A. and Herbst, E. (2007), «Moses: open source toolkit for statistical machine translation», in «Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions», ACL '07, pp. 177–180. (Cited on pages 346, 355, and 533.)

Koerich, A., Sabourin, R. and Suen, C. (2003), «Large vocabulary off-line handwriting recognition: A survey», *Pattern Analysis & Applications*, vol. 6 (2), pp. 97–121. (Cited on page 579.)

Kohonen, T. (1988), *Self-organization and associative memory*, vol. 8, Springer. (Cited on page 308.)

Kombrink, S., Hannemann, M. and Burget, L. (2012), «Out-of-vocabulary word detection and beyond», in «Detection and Identification of Rare Audiovisual Cues», pp. 57–65, Springer. (Cited on page 633.)

Konig, Y., Bourlard, H. and Morgan, N. (1996), «REMAP: Recursive Estimation and Maximization of A Posteriori Probabilities — Application to Transition-Based Connectionist Speech Recognition», in Touretzky, D. S., Mozer, M. C. and Hasselmo, M. E., eds., «Advances in Neural Information Processing Systems (NIPS)», vol. 8, pp. 388–394. (Cited on page 3.)

Kozielski, M., Forster, J. and Ney, H. (2012), «Moment-based Image Normalization for Handwritten Text Recognition», in «Frontiers in Handwriting Recognition (ICFHR), International Conference on», pp. 256–261, IEEE. (Cited on pages 296, 297, 558, 559, and 565.)

Krüger, S. E., Schafföner, M., Katz, M., Andelic, E. and Wendemuth, A. (2005), «Speech recognition with support vector machines in a hybrid system», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 993–996. (Cited on page 308.)

Kschischang, F. R., Frey, B. J. and Loeliger, H.-A. (2001), «Factor graphs and the sum-product algorithm», *Information Theory, IEEE Transactions on*, vol. 47 (2), pp. 498–519. (Cited on pages 115, 122, and 125.)

Kuhn, A. (2005), «Using Local Slant Correction to Normalize Handwritten Text Samples», . (Cited on pages 556 and 557.)

Kuhn, R. and De Mori, R. (1990), «A Cache-Based Natural Language Model for Speech Recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 12 (6), pp. 570–583. (Cited on pages 257 and 260.)

Kuich, W. (1997), *Semirings and formal power series: Their relevance to formal languages and automata*, Springer. (Cited on page 172.)

Kuno, S. and Oetinger, A. (1962), «multiple path syntactic analizer», *Information Processing*, pp. 306–311. (Cited on page 226.)

Kuo, H.-K. J., Fosler-Lussier, E., Jiang, H. and Lee, C.-H. (2002), «Discriminative training of language models for speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. I–325 –I–328. (Cited on pages 247 and 282.)

Kwisthout, J. (2009), *The Computational Complexity of Probabilistic Networks*, Ph.D. thesis, Universiteit Utrecht. (Cited on page 120.)

Laface, P., Vair, C. and Fissore, L. (1995), «A fast segmental Viterbi algorithm for large vocabulary recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 560–563. (Cited on page 424.)

Lafferty, J. D., McCallum, A. and Pereira, F. C. N. (2001), «Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data», in «Proceedings of the international conference on Machine learning (ICML)», ICML '01, pp. 282–289. (Cited on pages 119 and 160.)

Lallican, P. M., Viard-gaudin, C. and Knerr, S. (2000), «From Off-Line To On-Line Handwriting Recognition», in «In Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition», pp. 303–312. (Cited on page 571.)

Lang, B. (1974), «Deterministic techniques for efficient non-deterministic parsers», *Automata, Languages and Programming*, pp. 255–269. (Cited on page 228.)

Lang, B. (1988), «Parsing incomplete sentences», in «Proceedings of the 12th conference on Computational linguistics - Volume 1», COLING '88, pp. 365–371. (Cited on pages 175, 232, and 236.)

Lang, B. (1994), «Recognition can be Harder then Parsing», *Computational Intelligence — Intelligence Informatique*. (Cited on pages 175, 232, and 236.)

Lange, M. and Leiss, H. (2009), «To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm», *Informatica Didactica*, vol. 8, pp. 2008–2010. (Cited on pages 182, 191, 192, 193, and 227.)

Laporte, E. (1996), «Context-Free Parsing With Finite-State Transducers», in «In String Processing Colloquium», pp. 171–182. (Cited on page 231.)

Larson, M. and Jones, G. J. (2012), «Spoken content retrieval: A survey of techniques and technologies», *Foundations and Trends in Information Retrieval*, vol. 5 (4-5), pp. 235–422. (Cited on page 70.)

Layton, M. (2006), *Augmented statistical models for classifying sequence data*, Ph.D. thesis, Ph. D. thesis, Cambridge University. (Cited on page 164.)

Le, H.-S., Oparin, I., Allauzen, A., Gauvain, J.-L. and Yvon, F. (2011), «Structured Output Layer neural network language model», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 5524–5527. (Cited on pages 265, 369, and 379.)

Lecouteux, B., Linares, G., Esteve, Y. and Gravier, G. (2008), «Generalized driven decoding for speech recognition system combination», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 1549–1552. (Cited on pages 10, 274, and 275.)

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998), «Gradient-Based Learning Applied to Document Recognition», *Proceedings of the IEEE*, vol. 86 (11), pp. 2278–2324. (Cited on pages 154, 157, 160, 162, and 566.)

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M. and Huang, F. (2006), «A tutorial on energy-based learning», *Predicting Structured Data*. (Cited on page 117.)

Lee, A., Kawahara, T. and Doshita, S. (1998), «An efficient two-pass search algorithm using word trellis index.», in «International Conference on Spoken Language Processing (ICSLP)», vol. 98, pp. 1831–1834. (Cited on page 77.)

Lee, E. A. and Parks, T. M. (1995), «Dataflow process networks», vol. 83, pp. 773–801. (Cited on page 324.)

Lee, K. F. (1988), *Large-vocabulary speaker-independent continuous speech recognition: The Sphinx system*, Ph.D. thesis, Carnegie Mellon University. (Cited on pages 97 and 286.)

Lee, K.-F. (1990), «Context-dependent phonetic hidden Markov models for speaker-independent continuous speech recognition», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 38 (4), pp. 599–609. (Cited on page 44.)

Lee, K.-F., Hon, H.-W. and Reddy, R. (1990), «An overview of the SPHINX speech recognition system», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 38 (1), pp. 35–45. (Cited on page 297.)

Lee, L. and Rose, R. (1998), «A frequency warping approach to speaker normalization», *Speech and Audio Processing, IEEE Transactions on*, vol. 6 (1), pp. 49–60. (Cited on page 47.)

Lee, S.-W. and Kim, S.-Y. (1999), «Integrated segmentation and recognition of handwritten numerals with cascade neural network», *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 29 (2), pp. 285–290. (Cited on page 157.)

Leermakers, R. (1989), «How to cover a grammar», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», pp. 135–142, ACL. (Cited on pages 190, 191, and 227.)

Li, X. and Zhao, Y. (2007), «A fast and memory-efficient N-gram language model lookup method for large vocabulary continuous speech recognition», *Computer Speech and Language*, vol. 21 (1), pp. 1–25. (Cited on page 357.)

Lieb, R. (2006), *Efficient Integration of Hierarchical Knowledge Sources and the Estimation of Semantic Confidences for Automatic Speech Interpretation*, Ph.D. thesis, Technische Universität München, Munich, Germany. (Cited on page 346.)

Lifchitz, A., Maire, F. and Revuz, D. (2006a), «Fast Lexically Constrained Viterbi Algorithm (FLCVA): Simultaneous Optimization of Speed and Memory», *Computing Research Repository (CoRR)*. (Cited on page 415.)

Lifchitz, A., Maire, F. D. and Revuz, D. (2006b), «An Optimal Path Coding System for DAWG Lexicon-HMM», *Proc. of European Signal Processing Conference (EUSIPCO)*. (Cited on page 404.)

Linarès, G., Nocera, P., Massonié, D. and Matrouf, D. (2007), «The LIA speech recognition system: from 10xRT to 1xRT», in «TSD'07: Proceedings of the 10th international conference on Text, speech and dialogue», pp. 302–308. (Cited on page 313.)

Liu, X., Gales, M. and Woodland, P. (2012), «Paraphrastic Language Models», *Conference of the International Speech Communication Association (Interspeech)*. (Cited on page 269.)

Liu, X., Gales, M. and Woodland, P. (2013), «Use of contexts in language model interpolation and adaptation», *Computer Speech and Language*, vol. 27 (1), pp. 301–321. (Cited on page 258.)

Liu, X., Wang, Y., Chen, X., Gales, M. and Woodland, P. (2014), «Efficient lattice rescoring using recurrent neural network language models», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 4908–4912. (Cited on page 279.)

Liwicki, M. and Bunke, H. (2005), «IAM-OnDB - an on-line English sentence database acquired from handwritten text on a whiteboard», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 956–961. (Cited on pages 571, 609, 612, 646, 647, and 649.)

Liwicki, M., Bunke, H. *et al.* (2006a), «HMM-based on-line recognition of handwritten whiteboard notes», in «International Workshop on Frontiers in Handwriting Recognition (IWFHR)», pp. 595–599. (Cited on pages 570, 571, and 573.)

Liwicki, M., Scherz, M. and Bunke, H. (2006b), «Word Extraction from On-Line Handwritten Text Lines», in «International Conference on Pattern Recognition (ICPR)», vol. 2, pp. 929–933. (Cited on page 563.)

Llorens, D. and Casacuberta, F. (1999), «An experimental study of Histogram Pruning in Speech Recognition», in «VIII Congreso de la AERFAI»vol. II. (Cited on page 419.)

Llorens, D., Prat, F., Marzal, A., Vilar, J., Castro, M., Amengual, J., Barrachina, S., Castellanos, A., Espana, S., Gómez, J., Gorbe, J., Gordo, A., Palazón, V., Peris, G., Ramos-Garijo, R. and Zamora, F. (2008), «The UJIpenchars Database: A Pen-Based Database of Isolated Handwritten Characters», in «Proceedings of the International Conference on Language Resources and Evaluation (LREC)»European Language Resources Association (ELRA). (Cited on pages 639 and 646.)

Llorens Piñana, D. (2000), *Suavizado de automatas y traductores finitos estocasticos.*, Ph.d. thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. (Cited on pages 265, 266, 345, 359, and 516.)

Loeliger, H.-A. (2004), «An introduction to factor graphs», *Signal Processing Magazine, IEEE*, vol. 21 (1), pp. 28–41. (Cited on pages 116, 118, 122, and 125.)

Lopez, A. (2008), «Statistical Machine Translation.», *ACM Computing Surveys*, vol. 40 (3), pp. 1–49. (Cited on page 31.)

Lowerre, B. T. (1976), *The HARPY Speech Recognition System*, Ph.D. thesis, Carnegie Mellon University. (Cited on page 417.)

Lucchesi, C. L. and Kowaltowski, T. (1993), «Applications of finite automata representing large vocabularies», *Software: Practice and Experience*, vol. 23 (1), pp. 15–30. (Cited on page 414.)

Luján-Mares, M., Tamarit, V., Alabau, V., Martínez-Hinarejos, C.-D., Pastor, M., Sanchis, A. and Toselli, A. (2008), «iATROS: A speech and handwritting recognition system.», in «V Jornadas en Tecnologías del Habla (VJTH'2008)», pp. 75–78. (Cited on pages 421 and 453.)

Lundsteen, C. and Piper, J. (1989), *Automation of Cytogenetics*, Springer Verlag Berlin. (Cited on page 3.)

Lyngsø, R. B. and Pedersen, C. N. (2002), «The consensus string problem and the complexity of comparing hidden Markov models», *Journal of Computer and System Sciences*, vol. 65 (3), pp. 545–569. (Cited on page 6.)

Maas, A. L., Miller, S. D., O'Neil, T. M., Ng, A. Y. and Nguyen, P. (2012), «Word-level Acoustic Modeling with Convolutional Vector Regression», . (Cited on pages 52 and 256.)

MacKay, D. J. C. (2003), *Information Theory, Inference, and Learning Algorithms*. (Cited on page 115.)

MacKenzie, I. S., Kober, H., Smith, D., Jones, T. and Skepner, E. (2001), «LetterWise: prefix-based disambiguation for mobile text input», in «Proceedings of the 14th annual ACM symposium on User interface software and technology», UIST '01, pp. 111–120. (Cited on page 35.)

Madhvanath, S. and Govindaraju, V. (2001), «The Role of Holistic Paradigms in Handwritten Word Recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 23 (2), pp. 149–164. (Cited on pages 60 and 606.)

Madsen, O. L. and Kristensen, B. B. (1975), «On extended context free grammars and LR-parsing», *DAIMI Report Series*, vol. 4 (53). (Cited on page 179.)

Mahadevan, U. and Nagabushnam, R. (1995), «Gap metrics for word separation in handwritten lines», in «International Conference on Document Analysis and Recognition (ICDAR)», vol. 1, pp. 124–127. (Cited on page 567.)

Mahajan, M., Beeferman, D. and Huang, X. (1999), «Improved topic-dependent language modeling using information retrieval techniques», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 541–544. (Cited on page 259.)

Maire, F., Wathne, F. and Lifchitz, A. (2003), «Reduction of Non Deterministic Automata for Hidden Markov Model Based Pattern Recognition Applications», in «AI 2003: Advances in Artificial Intelligence», vol. 2903 of *Lecture Notes in Computer Science*, pp. 466–476. (Cited on page 414.)

Mangu, L., Brill, E. and Stolcke, A. (2000), «Finding consensus in speech recognition: word error minimization and other applications of confusion networks», *Computer Speech and Language*, vol. 14 (4), pp. 373–400. (Cited on pages 78 and 393.)

Manguel, A. (1998), *Une histoire de la lecture*, Actes Sud. (Cited on pages 1 and 4.)

Manmatha, R. and Rothfeder, J. (2005), «A scale space approach for automatically segmenting words from historical handwritten documents», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 27 (8), pp. 1212–1225. (Cited on page 567.)

Marinai, S., Gori, M. and Soda, G. (2005), «Artificial neural networks for document analysis and recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 27 (1), pp. 23–35. (Cited on pages 542 and 573.)

Mariño, J., Nogueiras, A. and Bonafonte, A. (1997), «The demiphone: an efficient subword unit for continuous speech recognition», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 1215–1218. (Cited on pages 44 and 313.)

Marti, U. and Bunke, H. (1999), «A full English sentence database for off-line handwriting recognition», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 705–708. (Cited on page 647.)

Marti, U.-V. and Bunke, H. (2000), «Handwritten sentence recognition», in «International Conference on Pattern Recognition (ICPR)», vol. 3, pp. 463–466. (Cited on pages 564 and 565.)

Marti, U.-V. and Bunke, H. (2001), «Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition systems», *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 15 (1), pp. 65–90. (Cited on pages 540 and 579.)

Marti, U.-V. and Bunke, H. (2002), «The IAM-database: an English sentence database for offline handwriting recognition», *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 5 (1), pp. 39–46. (Cited on pages 526, 579, 645, 646, 647, and 649.)

Martin, S. C., Liermann, J. and Ney, H. (1997), «Adaptive Topic-Dependent Language Modelling Using Word-Based Varigrams», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 1447–1450. (Cited on page 259.)

Martínez-Morató, J. (2011), *Aplicación para la supervisión en el preproceso de imágenes para escritura manuscrita*, Master's thesis, Facultad de Informàtica. Universidad Politècnica de Valencia, supervised by Salvador España and Francisco Zamora. (Cited on pages 575 and 639.)

Massonie, D., Nocera, P. and Linares, G. (2005), «Scalable language model look-ahead for LVCSR», in «Conference of the International Speech Communication Association (Interspeech)», pp. 569–572. (Cited on pages 357 and 431.)

McAllester, D., Collins, M. and Pereira, F. (2004), «Case-factor diagrams for structured probabilistic modeling», in «Proceedings of the 20th conference on Uncertainty in artificial intelligence», UAI '04, pp. 382–391. (Cited on page 148.)

McCallum, A., Freitag, D. and Pereira, F. C. N. (2000), «Maximum Entropy Markov Models for Information Extraction and Segmentation», in «ICML», pp. 591–598. (Cited on page 288.)

McCallum, A., Schultz, K. and Singh, S. (2009), «Factorie: Probabilistic programming via imperatively defined factor graphs», in «Advances in Neural Information Processing Systems (NIPS)», pp. 1249–1257. (Cited on page 127.)

Melibari, M., Poupart, P. and Lank, E. (2013), «Dynamic Sum-Product Networks», Tech. rep., University of Waterloo. (Cited on page 129.)

Menéndez-Pidal, X., Patrikar, A., Olorenshaw, L. and Honda, H. (2007), «Development of the compact English LVCSR acoustic model for embedded entertainment robot applications», *International Journal of Speech Technology*, vol. 10 (2-3), pp. 63–74. (Cited on pages 44 and 313.)

Michie, D. (1968), «Memo functions and machine learning», *Nature*, vol. 218 (5136), pp. 19–22. (Cited on pages 226 and 370.)

Miguel, A., Lleida, E., Rose, R., Buera, L., Saz, O. and Ortega, A. (2008), «Capturing Local Variability for Speaker Normalization in Speech Recognition», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16 (3), pp. 578 –593. (Cited on pages 9, 47, 53, and 139.)

Mikolov, T., Kombrink, S., Burget, L., Cernocky, J. and Khudanpur, S. (2011), «Extensions of recurrent neural network language model», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 5528–5531. (Cited on page 266.)

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013), «Efficient Estimation of Word Representations in Vector Space», *Computing Research Repository (CoRR)*, vol. abs/1301.3781. (Cited on page 371.)

Minka, T. and Winn, J. (2008), «Gates», in «Advances in Neural Information Processing Systems (NIPS)», pp. 1073–1080. (Cited on pages 109, 127, and 148.)

Mirowski, P. and LeCun, Y. (2009), «Dynamic Factor Graphs for Time Series Modeling», in Buntine, W., Grobelnik, M., Mladenic, D. and Shawe-Taylor, J., eds., «Machine Learning and Knowledge Discovery in Databases», vol. 5782 of *Lecture Notes in Computer Science*, pp. 128–143. (Cited on page 129.)

Mnih, A. and Teh, Y. W. (2012), «A fast and simple algorithm for training neural probabilistic language models», in «Proceedings of the international conference on Machine learning (ICML)», pp. 1751–1758. (Cited on page 370.)

Mohamed, A.-r., Sainath, T. N., Dahl, G., Ramabhadran, B., Hinton, G. E., Picheny, M. *et al.* (2011), «Deep belief networks using discriminative features for phone recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 5060–5063. (Cited on page 311.)

Mohri, M. (2002), «Semiring frameworks and algorithms for shortest-distance problems», *Journal of Automata, Languages and Combinatorics*, vol. 7 (3), pp. 321–350. (Cited on page 7.)

Mohri, M. (2003), «Edit-Distance of Weighted Automata», in Champarnaud, J.-M. and Maurel, D., eds., «Implementation and Application of Automata», vol. 2608 of *Lecture Notes in Computer Science*, pp. 1–23. (Cited on pages 272 and 406.)

Mohri, M. (2009), «Weighted automata algorithms», in «Handbook of weighted automata», pp. 213–254. (Cited on page 205.)

Mohri, M., Nederhof, M. *et al.* (2001), «Regular approximation of context-free grammars through transformation», *Robustness in language and speech technology*, vol. 17, pp. 153–163. (Cited on pages 193 and 245.)

Mohri, M., Pereira, F. and Riley, M. (1998), «A rational design for a weighted finite-state transducer library», in Wood, D. and Yu, S., eds., «Automata Implementation», vol. 1436 of *Lecture Notes in Computer Science*, pp. 144–158. (Cited on pages 271, 273, and 346.)

Mohri, M., Pereira, F. and Riley, M. (2002), «Weighted finite-state transducers in speech recognition», *Computer Speech and Language*, vol. 16 (1), pp. 69–88. (Cited on pages 174 and 185.)

Mohri, M., Pereira, F. and Riley, M. (2008), «Speech recognition with weighted finite-state transducers», *Handbook on Speech Processing and Speech Communication*. (Cited on page 173.)

Mohri, M. and Riley, M. (1998), «Network Optimizations for Large Vocabulary Speech Recognition», *Speech Communication*, vol. 25. (Cited on page 386.)

Moore, R. C. (2000), «Removing left recursion from context-free grammars», in «Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference», pp. 249–255, ACL. (Cited on page 182.)

Morgan, N. and Fosler-Lussier, E. (1998), «Combining multiple estimators of speaking rate», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 2, pp. 729–732. (Cited on pages 47, 54, and 103.)

Morin, F. and Bengio, Y. (2005), «Hierarchical probabilistic neural network language model», in «AISTATS'05», pp. 246–252. (Cited on page 379.)

Mozes, S., Weimann, O. and Ziv-Ukelson, M. (2007), «Speeding up HMM decoding and training by exploiting sequence repetitions», in «Combinatorial Pattern Matching», pp. 4–15, Springer. (Cited on page 436.)

Murphy, K. (2002), *Dynamic bayesian networks: Representation, inference and learning*, Ph.D. thesis, PhD thesis, University of California, Berkley, USA. (Cited on pages 129 and 149.)

Murphy, K. (2012), *Machine learning: a probabilistic perspective*, The MIT Press. (Cited on pages 106 and 114.)

Myers, C. and Rabiner, L. (1981), «A level building dynamic time warping algorithm for connected word recognition», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 29 (2), pp. 284–297. (Cited on page 7.)

Nadeu, C., Mariño, J. B., Hernando, J. and Nogueiras, A. (1996), «Frequency and time filtering of filter-bank energies for HMM speech recognition», in «International Conference on Spoken Language Processing (ICSLP)», vol. 1, pp. 430–433. (Cited on pages 12, 46, 517, and 626.)

Nair, N. U. and Sreenivas, T. (2010), «Joint evaluation of multiple speech patterns for speech recognition and training», *Computer Speech and Language*, vol. 24 (2), pp. 307–340. (Cited on page 80.)

Nair, V. and Hinton, G. E. (2010), «Rectified linear units improve restricted boltzmann machines», in «Proceedings of the international conference on Machine learning (ICML)», pp. 807–814. (Cited on page 598.)

Nanavati, A. and Rajput, N. (2006), «Improving Perplexity Measures To Incorporate Acoustic Confusability», in «Conference of the International Speech Communication Association (Interspeech)», pp. 17–21. (Cited on page 249.)

Natarajan, P., Saleem, S., Prasad, R., MacRostie, E. and Subramanian, K. (2008), «Multi-lingual Offline Handwriting Recognition Using Hidden Markov Models: A Script-Independent Approach», in «Arabic and Chinese Handwriting Recognition», pp. 231–250. (Cited on page 596.)

Nederhof, M. and Satta, G. (2003), «Probabilistic parsing as intersection», in «8th International Workshop on Parsing Technologies», pp. 137–148. (Cited on pages 186, 194, 231, 236, and 243.)

Nederhof, M.-J. (1993), «Generalized left-corner parsing», in «Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics», pp. 305–314, ACL. (Cited on page 228.)

Nederhof, M.-J. (2000), «Practical experiments with regular approximation of context-free languages», *Computational Linguistics*, vol. 26 (1), pp. 17–44. (Cited on pages 191, 193, and 387.)

Nederhof, M.-J. (2003), «Weighted deductive parsing and Knuth's algorithm», *Computational Linguistics*, vol. 29 (1), pp. 135–143. (Cited on page 230.)

Nederhof, M.-J. (2005), «A General Technique to Train Language Models on Language Models», *Computational Linguistics*, vol. 31 (2), pp. 173–186. (Cited on page 279.)

Nederhof, M.-J. and Satta, G. (2008), «Probabilistic Parsing», in Bel-Enguix, G., Jiménez-López, M. and Martín-Vide, C., eds., «New Developments in Formal Languages and Applications», vol. 113 of *Studies in Computational Intelligence*, pp. 229–258. (Cited on pages 180 and 231.)

Ney, H. (1991), «Speech recognition in a neural network framework: discriminative training of Gaussian models and mixture densities as radial basis functions», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 573–576. (Cited on page 307.)

Ney, H., Ortmanns, S. and Lindam, I. (1997), «Extensions to the Word Graph Method for Large Vocabulary Continuous Speech Recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 1791–1794. (Cited on page 494.)

Ney, H., Haeb-Umbach, R., Tran, B.-H. and Oerder, M. (1992), «Improvements in beam search for 10000-word continuous speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 9–12. (Cited on pages 409 and 412.)

Ney, H., Niessen, S., Och, F., Sawaf, H., Tillmann, C. and Vogel, S. (2000), «Algorithms for statistical translation of spoken language», *Speech and Audio Processing, IEEE Transactions on*, vol. 8 (1), pp. 24–36. (Cited on page 32.)

Nguyen, P. (2002), *Speaker Adaptation: Modeling Variabilities*, Ph.D. thesis, École Polytechnique Fédéral de Laussane. (Cited on pages 404 and 453.)

Nguyen, P., Rigazio, L. and Junqua, J.-C. (2000), «EWAVES: An efficient decoding algorithm for lexical tree based speech recognition», in «International Conference on Spoken Language Processing (ICSLP)», vol. 4, pp. 286–289. (Cited on pages 453, 459, and 465.)

Niblack, W. (1990), *An Introduction to Digital Image Processing*, Prentice-Hall, Inc. (Cited on page 542.)

Nielsen, J. and Sand, A. (2011), «Algorithms for a Parallel Implementation of Hidden Markov Models with a Small State Space», in «Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on», pp. 452–459. (Cited on pages 389, 390, and 436.)

Nolden, D., Ney, H. and Schlüter, R. (2010), «Time Conditioned Search in Automatic Speech Recognition Reconsidered», in «Conference of the International Speech Communication Association (Interspeech)». (Cited on pages 432 and 493.)

Nolden, D., Ney, H. and Schluter, R. (2011a), «Exploiting sparseness of backing-off language models for efficient look-ahead in LVCSR», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 4684–4687. (Cited on pages 277 and 431.)

Nolden, D., Schlüter, R. and Ney, H. (2011b), «Acoustic Look-Ahead for More Efficient Decoding in LVCSR», in «Twelfth Annual Conference of the International Speech Communication Association». (Cited on page 422.)

Nolden, D., Schluter, R. and Ney, H. (2013), «Efficient nearly error-less LVCSR decoding based on incremental forward and backward passes», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 66–71. (Cited on pages 418 and 426.)

Novak, M., Hampl, R., Krbec, P., Bergl, V. and Sedivy, J. (2003), «Two-pass search strategy for large list recognition on embedded speech recognition platforms», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. I–200. (Cited on pages 414 and 452.)

Ntirogiannis, K., Gatos, B. and Pratikakis, I. (2013), «Performance evaluation methodology for historical document image binarization», *Image Processing, IEEE Transactions on*, vol. 22 (2), pp. 595–609. (Cited on page 544.)

Och, F. (2003), «Minimum Error Rate Training in Statistical Machine Translation», in «Proc. of ACL», pp. 160–167. (Cited on pages 246, 583, and 611.)

Och, F. J. and Ney, H. (2003), «A systematic comparison of various statistical alignment models», *Computational Linguistics*, vol. 29 (1), pp. 19–51. (Cited on pages 102 and 533.)

Och, F. J. and Ney, H. (2004), «The Alignment Template Approach to Statistical Machine Translation», *Computational Linguistics*, vol. 30, pp. 417–449. (Cited on pages 31 and 32.)

Odell, J. J. (1995), *The Use of Context in Large Vocabulary Speech Recognition*, Ph.D. thesis, Queen's College. (Cited on pages 296 and 298.)

Oerder, M. and Ney, H. (1993), «Word Graphs: An Efficient Interface Between Continuous Speech Recognition And Language Understanding», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 2, pp. 119–122. (Cited on pages 482 and 487.)

Oger, S. and Linarès, G. (2014), «Web-based possibilistic language models for automatic speech recognition», *Computer Speech & Language*, vol. 28 (4), pp. 923–939. (Cited on page 282.)

Okanohara, D. and Tsujii, J. (2007), «A discriminative language model with pseudo-negative samples», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», vol. 45, p. 73. (Cited on page 282.)

Okasaki, C. (1995), «Simple and efficient purely functional queues and deques», *Journal of Functional Programming*, vol. 5 (4), pp. 583–592. (Cited on page 391.)

Oncina, J. (2009), «Optimum algorithm to minimize human interactions in sequential Computer Assisted Pattern Recognition», *Pattern Recognition Letters*, vol. 30 (5), pp. 558–563. (Cited on pages 8 and 36.)

Ortmanns, S. (1998), *Effiziente Suchverfahren zur Erkennung kontinuierlich gesprochener Sprache*, Ph.D. thesis, RWTH Aachen University. (Cited on pages 419, 453, 454, and 494.)

Ortmanns, S., Eiden, A. and Ney, H. (1998), «Improved lexical tree search for large vocabulary speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 2, pp. 817–820. (Cited on page 433.)

Ortmanns, S. and Ney, H. (2000), «The time-conditioned approach in dynamic programming search for LVCSR», *Speech and Audio Processing, IEEE Transactions on*, vol. 8 (6), pp. 676–687. (Cited on pages 165, 396, 476, 483, and 505.)

Ortmanns, S., Ney, H., Eiden, A. and Coenen, N. (1996), «Look-ahead techniques for improved beam search», in «Proc. CRIM-FORWISS Workshop», pp. 10—-22. (Cited on pages 277, 422, and 431.)

Ortmanns, S., Reichl, W., Chou, W. and Lee, C.-H. (1999), «An Efficient Decoding Approach for Dialogue Systems», in «ESCA Tutorial and Research Workshop (ETRW) on Interactive Dialogue in Multi-Modal Systems». (Cited on pages 484, 493, and 625.)

Ostendorf, M. (1999), «Moving beyond the 'beads-on-a-string'model of speech», pp. 79–84. (Cited on pages 5 and 130.)

Ostendorf, M., Digalakis, V. and Kimball, O. (1996), «From HMM's to segment models: a unified view of stochastic modeling for speech recognition», *IEEE Trans. Speech and Audio Processing*, vol. 4 (5), pp. 360–378. (Cited on pages 150, 292, 300, and 304.)

Otsu, N. (1979), «A threshold selection method from gray-level histograms», *IEEE Trans. on Systems, Man and Cybernetics*, vol. 9 (1), pp. 62–66. (Cited on page 542.)

Palacios-Corella, A., Zamora-Martínez, F., España-Boquera, S. and Castro-Bleda, M. (2014), «First steps towards Skipping NNLMs», in «VIII Jornadas en Tecnología del Habla and IV Iberian SLTech Workshop (IberSPEECH)», pp. 111–118. (Cited on pages 373, 530, and 638.)

Paliwal, K. (1993), «Use of temporal correlation between successive frames in a hidden Markov model based speech recognizer», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 2, pp. 215–218. (Cited on page 290.)

Papavassiliou, V., Stafylakis, T., Katsouros, V. and Carayannis, G. (2010), «Handwritten document image segmentation into text lines and words», *Pattern Recognition*, vol. 43, pp. 369–377. (Cited on pages 567 and 568.)

Papineni, K., Roukos, S., Ward, T. and Zhu, W.-J. (2002), «BLEU: a method for automatic evaluation of machine translation», in «Proceedings of the 40th annual meeting on association for computational linguistics», pp. 311–318, Association for Computational Linguistics. (Cited on page 98.)

Parada, M. C. (2011), *Learning Sub-Word Units and Exploiting Contextual Information for Open Vocabulary Speech Recognition*, Ph.D. thesis, Johns Hopkins University, Baltimore, Maryland. (Cited on page 270.)

Pastor, M. and Paredes, R. (2010), «Bi-modal Handwritten Text Recognition (BiHTR) ICPR 2010 Contest Report», in Ünay, D., Çataltepe, Z. and Aksoy, S., eds., «Recognizing Patterns in Signals, Speech, Images and Videos», vol. 6388 of *Lecture Notes in Computer Science*, pp. 1–13, Springer Berlin / Heidelberg. (Cited on pages 610 and 649.)

Pastor, M., Toselli, A. and Vidal, E. (2004), «Projection profile based algorithm for slant removal», in «Proc. International Conference on Image Analysis and Recognition», vol. 3212 of *Lecture Notes in Computer Science*, pp. 183–190. (Cited on pages 555 and 557.)

Pastor, M., Toselli, A. and Vidal, E. (2005), «Writing speed normalization for on-line handwritten text recognition», in «International Conference on Document Analysis and Recognition (ICDAR)», vol. 2, pp. 1131–1135. (Cited on page 570.)

Pastor, M., Vidal, E. and Casacuberta, F. (2009), «A bi-modal handwritten text corpus», Tech. rep., Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Spain. (Cited on pages 571, 609, 610, and 649.)

Pastor, M., Toselli, A. H., Casacuberta, F. and Vidal, E. (2010), «A Bi-modal Handwritten Text Corpus: Baseline Results», in «International Conference on Pattern Recognition (ICPR)», pp. 1933–1936. (Cited on page 610.)

Pastor Gadea, M. (2007), *Aportaciones al Reconocimiento Automático de Texto Manuscrito*, Ph.d. thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, advisors: Dr. Enrique Vidal Ruiz and Dr. Alejandro H. Toselli. (Cited on pages 544, 546, 552, 579, 582, 583, 586, 588, 589, 611, and 626.)

Pastor-Pellicer, J., Zamora-Martínez, F., España-Boquera, S. and Castro-Bleda, M.-J. (2013), «F-Measure as the Error Function to Train Neural Networks», in «Advances in Computational Intelligence», vol. 7902 of *Lecture Notes in Computer Science*, pp. 376–384, Springer. (Cited on pages 574, 606, and 638.)

Pastor-Pellicer, J., España-Boquera, S., Zamora-Martínez, F. and Castro-Bleda, M. J. (2014), «Handwriting Normalization by Zone Estimation using HMM/ANNs», *Frontiers in Handwriting Recognition (ICFHR), International Conference on*. (Cited on pages 597, 635, and 640.)

Pastor-Pellicer, J., España-Boquera, S., Castro-Bleda, M. J. and Zamora-Martínez, F. (2015a), «A combined Convolutional Neural Network and Dynamic Programming approach for text line normalization», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. –. (Cited on pages 574, 597, 598, 635, and 639.)

Pastor-Pellicer, J., Garz, A., Ingold, R. and Castro-Bleda, M. J. (2015b), «Combining Learned Script Points and Combinatorial Optimization for Text Line Extraction», in «Historical Document Image and Processing (HIP 2015), 3rd International Workshop on», pp. –. (Cited on page 628.)

Pastor-Pellicer, J., España-Boquera, S., Zamora-Martínez, F., Afzal, M. Z. and Castro-Bleda, M. J. (2015c), «Insights on the Use of Convolutional Neural Networks for Document Image Binarization», in «Advances in Computational Intelligence», pp. 115–126, Springer International Publishing. (Cited on pages 574, 597, and 638.)

Patel, S. (1995), «A lower-complexity Viterbi algorithm», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 592–595. (Cited on pages 383 and 437.)

Patel, S. (1997), «An O(N sqrt(E)) Viterbi algorithm», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 3, pp. 1795–1798. (Cited on pages 383 and 438.)

Pauls, A. and Klein, D. (2011), «Faster and smaller N-gram language models», in «Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1», HLT '11, pp. 258–267. (Cited on page 356.)

Pereira, F. C. and Riley, M. D. (1997), «15 Speech Recognition by Composition of Weighted Finite Automata», *Finite-state language processing*, p. 431. (Cited on pages 201 and 205.)

Perl, Y., Itai, A. and Avni, H. (1978), «Interpolation search —- a log logN search», *Communications of the ACM*, vol. 21, pp. 550–553. (Cited on pages 357 and 361.)

Pesch, H., Hamdani, M., Forster, J. and Ney, H. (2012), «Analysis of Preprocessing Techniques for Latin Handwriting Recognition», in «Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on», pp. 280 –284. (Cited on page 542.)

Pfau, T., Faltlhauser, R. and Ruske, G. (2000), «A combination of speaker normalization and speech rate normalization for automatic speech recognition», in «International Conference on Spoken Language Processing (ICSLP)», vol. 4, pp. 362–365. (Cited on page 293.)

Phillips, S. and Rogers, A. (1999), «Parallel Speech Recognition», *Int. J. Parallel Program.*, vol. 27 (4), pp. 257–288. (Cited on page 406.)

Pieraccini, R. and Levin, E. (1992), «Stochastic representation of semantic structure for speech understanding», *Speech Communication*, vol. 11 (2-3), pp. 283–288. (Cited on page 33.)

Pieraccini, R., Levin, E. and Vidal, E. (1993), «Learning how to understand language», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 1407–1412. (Cited on page 33.)

Plahl, C. (2014), *Neural Network based Feature Extraction for Speech and Image Recognition*, Ph.D. thesis, RWTH Aachen University. (Cited on page 156.)

Plamondon, R. and Srihari, S. (2000), «Online and off-line handwriting recognition: a comprehensive survey», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 22 (1), pp. 63 –84. (Cited on pages 37 and 579.)

Plötz, T. and Fink, G. (2009), «Markov models for offline handwriting recognition: a survey», *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 12, pp. 269–298. (Cited on pages 2, 37, 566, and 596.)

Pouly, M. and Kohlas, J. (2012), *Generic Inference: A Unifying Theory for Automated Reasoning*, John Wiley & Sons. (Cited on pages 121 and 123.)

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P. *et al.* (2011), «The Kaldi speech recognition toolkit», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on». (Cited on pages 323, 346, 629, and 631.)

Prat, F., Aibar, P., Marzal, A. and Vidal, E. (1994), «El problema de la evaluación de un sistema de reconocimiento del habla mediante un único valor numérico», Tech. rep., DSIC, UPV, Spain. (Cited on page 97.)

Prieto, N. and Vidal, E. (1992), «Learning Language Models through the ECGI method», *Speech Communication*, vol. 11 (2), pp. 299–309. (Cited on page 266.)

Printz, H. and Olsen, P. (2002), «Theory and practice of acoustic confusability», *Computer Speech and Language*, vol. 16, pp. 131–164. (Cited on page 248.)

Pylkkönen, J. (2004), *Phone Duration Modeling Techniques in Continuous Speech Recognition*, Master's thesis, Helsinki University of Technology. (Cited on pages 48 and 295.)

Pylkkönen, J. (2005), «New Pruning Criteria for Efficient Decoding», in «Conference of the International Speech Communication Association (Interspeech)», pp. 581–584. (Cited on page 416.)

Pynadath, D. and Wellman, M. (1998), «Generalized queries on probabilistic context-free grammars», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 20 (1), pp. 65 –77. (Cited on page 148.)

Qing, G., Fang, Z., Jian, W. and Wenhu, W. (1999), «An new method used in HMM for modeling frame correlation», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 169–172. (Cited on page 290.)

Quint, J. (2004), «On the equivalence of weighted finite-state transducers», in «Proceedings of the ACL 2004 on Interactive poster and demonstration sessions»ACLdemo '04. (Cited on page 231.)

Rabiner, L. and Huang, B. H. (1993), *Fundamentals of Speech Recognition*, Prentice-Hall. (Cited on page 131.)

Rabiner, L. and Juang, B. (1986), «An introduction to hidden Markov models», *ASSP Magazine, IEEE*, vol. 3 (1), pp. 4–16. (Cited on pages 141, 145, and 295.)

Ragni, A. (2013), *Discriminative models for speech recognition*, Ph.D. thesis, University of Cambridge. (Cited on page 163.)

Ramshaw, L. A. and Marcus, M. P. (1995), «Text Chunking using Transformation-Based Learning», p. 5040. (Cited on page 30.)

Rashid, S., Shafait, F. and Breuel, T. (2012), «Scanning Neural Network for Text Line Recognition», in «10th IAPR Workshop on Document Analysis Systems, DAS'12», . (Cited on page 157.)

Ratinov, L. and Roth, D. (2009), «Design challenges and misconceptions in named entity recognition», in «Proceedings of the Thirteenth Conference on Computational Natural Language Learning», xu '09, pp. 147–155. (Cited on page 30.)

Ravi, S. and Knight, K. (2011), «Deciphering foreign language», in «Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1», HLT '11, pp. 12–21. (Cited on page 31.)

Ravishankar, M. K. (1996), *Efficient Algorithms for Speech Recognition*, Ph.D. thesis, Carnegie Mellon University. (Cited on pages 77, 363, 411, and 429.)

Renals, S., Hochberg, M. and of Sheffield. Department of Computer Science, U. (1995), «Decoder technology for connectionist large vocabulary speech recognition», Tech. Rep. CS-95-17, Dept. of Computer Science, University of Sheffield. (Cited on page 416.)

Riccardi, G., Bocchieri, E. and Pieraccini, R. (1995), «Non-deterministic stochastic language models for speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 237–240. (Cited on pages 265, 345, and 359.)

Riedhammer, K., Bocklet, T., Ghoshal, A. and Povey, D. (2012), «Revisiting semi-continuous hidden Markov models», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 4721–4724. (Cited on page 306.)

Rigoll, G. (1994), «Maximum mutual information neural networks for hybrid connectionist-HMM speech recognition systems», *Speech and Audio Processing, IEEE Transactions on*, vol. 2 (1), pp. 175–184. (Cited on page 308.)

Riley, M., Pereira, F. and Mohri, M. (1997), «Transducer Composition for Context-Dependent Network Expansion», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 1427–1430. (Cited on pages 345, 398, and 491.)

Roark, B., Sproat, R. and Shafran, I. (2011), «Lexicographic semirings for exact automata encoding of sequence models», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», HLT'11, pp. 1–5. (Cited on pages 173 and 360.)

Robinson, T., Hochberg, M. and Renals, S. (1996), «The use of recurrent neural networks in continuous speech recognition», *Automatic Speech and Speaker Recognition: Advanced Topics*, vol. 355, pp. 233–258. (Cited on page 308.)

Roche, E. (1994), «Two parsing algorithms by means of finite state transducers», in «Proceedings of the 15th conference on Computational linguistics - Volume 1», COLING '94, pp. 431–435. (Cited on page 231.)

Rodríguez, L., García-Varea, I., Revuelta-Martínez, A. and Vidal, E. (2010), «A multimodal interactive text generation system», in «International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction», ICMI-MLMI '10, pp. 11:1–11:2. (Cited on page 36.)

Rodriguez-Serrano, J. A., Gordo, A. and Perronnin, F. (2014), «Label Embedding: A Frugal Baseline for Text Recognition», *International Journal of Computer Vision*, pp. 1–15. (Cited on page 164.)

Rodríguez, L., Casacuberta, F. and Vidal, E. (2007), «Computer Assisted Transcription of Speech», in «Pattern Recognition and Image Analysis», vol. 4477 of *Lecture Notes in Computer Science*, pp. 241–248. (Cited on page 380.)

Romero, V. (2005), *Mejora de la normalización de tamaño de texto manuscrito off-line*, Master's thesis, Facultad de Informàtica. Universidad Politécnica de Valencia, supervised by Moisés Pastor. (Cited on page 562.)

Romero, V., FornéS, A., Serrano, N., SáNchez, J. A., Toselli, A. H., Frinken, V., Vidal, E. and LladóS, J. (2013), «The ESPOSALLES database: An ancient marriage license corpus for off-line handwriting recognition», *Pattern Recognition*, vol. 46 (6), pp. 1658–1669. (Cited on page 628.)

Rose, R. and Paul, D. (1990), «A hidden Markov model based keyword recognition system», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 129–132. (Cited on page 71.)

Rosenberg, M. S. (2009), *Sequence alignment: methods, models, concepts, and strategies*, Univ of California Press. (Cited on page 68.)

Rosenfeld, R. (1996), «A maximum entropy approach to adaptive statistical language modelling», *Computer Speech & Language*, vol. 10 (3), pp. 187–228. (Cited on pages 260 and 267.)

Rosenfeld, R. (1997), «A whole sentence maximum entropy language model», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 230–237. (Cited on page 244.)

Rosenkrantz, D. and Lewis, P. (1970), «Deterministic left corner parsing», in «Switching and Automata Theory, 1970., IEEE Conference Record of 11th Annual Symposium on», pp. 139–152. (Cited on page 228.)

Rosenkrantz, D. J. (1967), «Matrix Equations and Normal Forms for Context-Free Grammars», *J. ACM*, vol. 14 (3), pp. 501–507. (Cited on pages 182, 183, and 184.)

Rosti, A.-V. I. (2004), *Linear Gaussian models for speech recognition*, Ph.D. thesis, Cambridge University. (Cited on pages 300 and 303.)

Rottland, J. and Rigoll, G. (2000), «Tied posteriors: an approach for effective introduction of context dependency in hybrid NN/HMM LVCSR», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 3, pp. 1241–1244. (Cited on page 307.)

Ruiz-Pinales, J., Jaime-Rivas, R. and Castro-Bleda, M. J. (2007), «Holistic cursive word recognition based on perceptual features», *Pattern Recognition Letters*, vol. 28 (13), pp. 1600–1609. (Cited on pages 60 and 606.)

Russel, M., Ponting, K., Peeling, S. M., Browning, S. R., Bridle, J., Moore, R., Galiano, I. and Howell, P. (1990), «The ARM continuous speech recognition system», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 69–72. (Cited on page 297.)

Russell, M. and Cook, A. (1987), «Experimental evaluation of duration modelling techniques for automatic speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 12, pp. 2376–2379. (Cited on page 295.)

Rybach, D. (2014), *Investigations on Search Methods for Speech Recognition using Weighted Finite-State Transducers*, Ph.D. thesis, RWTH Aachen University. (Cited on pages 417 and 453.)

Rybach, D., Riley, M. and Alberti, C. (2013), «Direct construction of compact context-dependency transducers from data», *Computer Speech and Language*. (Cited on page 491.)

Rybach, D., Gollan, C., Heigold, G., Hoffmeister, B., Lööf, J., Schlüter, R. and Ney, H. (2009), «The RWTH Aachen University Open Source Speech Recognition System», in «Conference of the International Speech Communication Association (Interspeech)», pp. 2111–2114. (Cited on page 325.)

Saenger, P. (2000), *Space Between Words: The Origins of Silent Reading*, Cambridge University Press. (Cited on pages 1 and 28.)

Saers, M. and Wu, D. (2011), «Reestimation of reified rules in semiring parsing and biparsing», in «Proceedings of the Fifth Workshop on Syntax, Semantics and Structure in Statistical Translation», SSST-5, pp. 70–78. (Cited on page 237.)

Sag, I., Baldwin, T., Bond, F., Copestake, A. and Flickinger, D. (2002), «Multiword expressions: A pain in the neck for NLP», *Computational Linguistics and Intelligent Text Processing*, pp. 189–206. (Cited on page 252.)

Sagae, K., Christian, G., DeVault, D. and Traum, D. R. (2009), «Towards Natural Language Understanding of Partial Speech Recognition Results in Dialogue Systems.», in «Short Paper Proceedings of NAACL HLT». (Cited on page 8.)

Sainath, T. N., Kingsbury, B. and Ramabhadran, B. (2012), «Auto-encoder bottleneck features using deep belief networks», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 4153–4156. (Cited on page 311.)

Sakoe, H. (1979), «Two-level DP-matching–A dynamic programming-based pattern matching algorithm for connected word recognition», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 27 (6), pp. 588–595. (Cited on pages 142 and 163.)

Sánchez, J. A., Casacuberta, F., Aibar, P., Llorens, D. and Castro, M. J. (1999), «Fast phoneme look-ahead in the ATROS system», in «VIII Congreso de la AERFAI», vol. I, pp. 77–84. (Cited on page 422.)

Sanchez-Cortina, I., Serrano, N., Sanchis, A. and Juan, A. (2012), «A prototype for interactive speech transcription balancing error and supervision effort», in «Proceedings of the 2012 ACM international conference on Intelligent User Interfaces», IUI '12, pp. 325–326. (Cited on page 86.)

Sand, A., Kristiansen, M., Pedersen, C. N. and Mailund, T. (2013), «zipHMMlib: a highly optimised HMM library exploiting repetitions in the input to speed up the forward algorithm», *BMC bioinformatics*, vol. 14 (1), p. 339. (Cited on page 436.)

Sandve, G., Nekrutenko, A., Taylor, J. and Hovig, E. (2013), «Ten simple rules for reproducible computational research.», *PLoS computational biology*, vol. 9 (10). (Cited on pages 91 and 95.)

Sang, E. F. T. K. and Veenstra, J. (1999), «Representing text chunks», in «Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics», EACL '99, pp. 173–179. (Cited on page 30.)

Saon, G. and Padmanabhan, M. (2001), «Data-driven approach to designing compound words for continuous speech recognition», *Speech and Audio Processing, IEEE Transactions on*, vol. 9 (4), pp. 327 –332. (Cited on pages 252 and 604.)

Sarawagi, S. (2006), «Efficient inference on sequence segmentation models», in «Proceedings of the international conference on Machine learning (ICML)», pp. 793–800. (Cited on page 62.)

Sato, T. (2007), «Inside-outside probability computation for belief propagation», in «Proceedings of the 20th international joint conference on Artifical intelligence», IJCAI'07, pp. 2605–2610. (Cited on page 148.)

Sayre, K. M. (1973), «Machine recognition of handwritten words: a project report», *Pattern Recognition*, vol. 5 (3), pp. 213–228. (Cited on pages 2, 37, 74, and 577.)

Schalkwyk, J., Hetherington, L. and Story, E. (2003), «Speech recognition with dynamic grammars using finite-state transducers», in «European Conference on Speech Communication and Technology (Eurospeech)», pp. 1969–1972. (Cited on page 231.)

Schambach, M. P. (2009), «Recurrent HMMs and Cursive Handwriting Recognition Graphs», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 1146–1150. (Cited on pages 270 and 599.)

Scharenborg, O., ten Bosch, L. and Boves, L. (2007), *Robust Speech Recognition and Understanding*, chap. Early Decision Making in Continuous Speech, pp. 333–350. (Cited on page 9.)

Schlapbach, A. and Bunke, H. (2005), «Writer identification using an HMM-based handwriting recognition system: To normalize the input or not», in «Proc. 12th Conf. of the Int. Graphonomics Society», pp. 138–142. (Cited on pages 559 and 563.)

Schlüter, R. and Ney, I. H. (2010), «Advanced methods in Automatic Speech Recognition», URL `http://www-i6.informatik.rwth-aachen.de/web/Teaching/Lectures/SS10/advasr/slides/advasrSS10printable.pdf`. (Cited on pages 453 and 454.)

Schmidler, S. C., Liu, J. S. and Brutlag, D. L. (2004), «Stochastic Segment Interaction Models for Biological Sequence Analysis», *J. Amer. Statist. Assoc.*, (submitted to). (Cited on page 290.)

Schukat-Talamazzini, E., Niemann, H., Eckert, W., Kuhn, T. and Rieck, S. (1992), «Acoustic modelling of subword units in the Isadora speech recognizer», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 577 –580 vol.1. (Cited on page 45.)

Schulz, K. U. and Mihov, S. (2002), «Fast string correction with Levenshtein automata», *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 5 (1), pp. 67–85. (Cited on page 406.)

Schwartz, R. and Austin, S. (1991), «A comparison of several approximate algorithms for finding multiple (N-best) sentence hypotheses», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 701 –704 vol. 1. (Cited on page 428.)

Schwartzkopf, W. C. (2002), *Maximum Likelihood Techniques for Joint Segmentation-Classification of Multi-spectral Chromosome Images*, Ph.D. thesis, The University of Texas at Austin. (Cited on page 3.)

Schwenk, H. (2007), «Continuous space language models», *Comput. Speech Lang.*, vol. 21 (3), pp. 492–518. (Cited on pages 263 and 350.)

Science, R. C. and Ravishankar, M. K. (1993), «Parallel Implementation of Fast Beam Search for Speaker-Independent Continuous Speech Recognition», *Computer Science and Automation, IISc, India*. (Cited on page 412.)

Scott, E. and Johnstone, A. (2010), «GLL parsing», *Electronic Notes in Theoretical Computer Science*, vol. 253 (7), pp. 177–189. (Cited on page 228.)

Segarra, E. and Hurtado, L. (1997), «Construction of Language Models using the Morphic Generator Grammatical Inference (MGGI) Methodology», *European Conference on Speech Communication and Technology (Eurospeech)*, pp. 2695–2698. (Cited on pages 266 and 269.)

Seide, F. (2005), «The use of virtual hypothesis copies in decoding of large-vocabulary continuous speech», *Speech and Audio Processing, IEEE Transactions on*, vol. 13 (4), pp. 520–533. (Cited on page 428.)

Seiler, R., Schenkel, M. and Eggimann, F. (1996), «Off-line Cursive Handwriting Recognition Compared with On-line Recognition», in «International Conference on Pattern Recognition (ICPR)», pp. 505–509. (Cited on pages 546 and 547.)

Seni, G. and Cohen, E. (1994), «External word segmentation of off-line handwritten text lines», *Pattern Recognition*, vol. 27 (1), pp. 41–52. (Cited on page 567.)

Senior, A. W. and Robinson, A. J. (1998), «An off-line cursive handwritten recognition system», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 20 (3), pp. 309–321. (Cited on page 553.)

Serrano, N., Giménez, A., Civera, J., Sanchis, A. and Juan, A. (2013), «Interactive handwriting recognition with limited user effort», *International Journal of Document Analysis and Recognition (IJDAR)*, pp. 1–13. (Cited on pages 83 and 86.)

Settles, B. (2010), «Active learning literature survey», Technical Report 1648, University of Wisconsin, Madison. (Cited on pages 83 and 90.)

Shannon, B. J. and Paliwal, K. K. (2003), «A comparative study of filter bank spacing for speech recognition», in «Microelectronic engineering research conference»vol. 41. (Cited on page 46.)

Shapiro, L. G. and Stockman, G. C. (2002), *Computer Vision*, Prentice Hall. (Cited on page 568.)

Shenoy, P. P. (1992), «Fuzzy Logic for the Management of Uncertainty», chap. Valuation-based Systems: A Framework for Managing Uncertainty in Expert Systems, pp. 83–104. (Cited on pages 116 and 123.)

Shi, Q., Altun, Y., Smola, A. J. and Vishwanathan, S. (2007), «Semimarkov models for sequence segmentation», in «Proc. of Empirical Methods in Natural Language Processing (EMNLP)», pp. 640–648. (Cited on pages 28, 92, and 136.)

Shi, Y., Zhang, W.-Q., Cai, M. and Liu, J. (2014), «Efficient One-Pass Decoding with NNLM for Speech Recognition», *Signal Processing Letters, IEEE*, vol. 21 (4), pp. 377–381. (Cited on page 370.)

Shieber, S. M., Schabes, Y. and Pereira, F. C. (1995), «Principles and implementation of deductive parsing», *The Journal of logic programming*, vol. 24 (1), pp. 3–36. (Cited on pages 227 and 230.)

Sikkel, K. (1993), *Parsing Schemata*, Ph.D. thesis, University of Twente. (Cited on pages 227 and 230.)

Simard, P., Steinkraus, D. and Agrawala, M. (29 Aug.-1 Sept. 2005), «Ink normalization and beautification», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 1182–1187. (Cited on page 546.)

Singer, E. and Lippman, R. (1992), «A speech recognizer using radial basis function neural networks in an HMM framework», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 629–632 vol.1. (Cited on pages 307 and 308.)

Siniscalchi, S., Svendsen, T. and Lee, C.-H. (2013), «A Bottom-Up Modular Search Approach to Large Vocabulary Continuous Speech Recognition», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 21 (4), pp. 786–797. (Cited on page 506.)

Siu, M. and Chan, A. (2006), «A Robust Viterbi Algorithm Against Impulsive Noise With Application to Speech Recognition», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 14 (6), pp. 2122–2133. (Cited on pages 8 and 423.)

Siu, M. and Gish, H. (1999), «Evaluation of word confidence for speech recognition systems», *Computer Speech and Language*, vol. 13 (4), pp. 299–319. (Cited on page 102.)

Sixtus, A. (2003), *Across-Word Phoneme Models for Large Vocabulary Continuous Speech Recognition*, Ph.D. thesis, RWTH Aachen University. (Cited on pages 313, 397, 398, 399, 413, 414, 417, 419, 420, 430, 453, 454, and 493.)

Smith, R. N. (1973), «Automatic Steno translation», in «Proceedings of the ACM annual conference», ACM '73, pp. 92–96. (Cited on page 34.)

Snover, M., Dorr, B., Schwartz, R., Micciulla, L. and Makhoul, J. (2006), «A study of translation edit rate with targeted human annotation», in «Proceedings of association for machine translation in the Americas», pp. 223–231. (Cited on page 98.)

Soltau, H., Saon, G. and Kingsbury, B. (2010), «The IBM Attila speech recognition toolkit», in «Proc. IEEE Workshop on Spoken Language Technology». (Cited on pages 324 and 346.)

Soltau, H., Metze, F., Fügen, C. and Waibel, A. (2001), «A one-pass decoder based on polymorphic linguistic context assignment», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 214–217, IEEE. (Cited on pages 324, 344, and 427.)

Song, X., Ding, S. and Lin, C.-Y. (2008), «Better binarization for the CKY parsing», in «Proc. of Empirical Methods in Natural Language Processing (EMNLP)», pp. 167–176, ACL. (Cited on page 193.)

Sporleder, C. and Lapata, M. (2006), «Broad coverage paragraph segmentation across languages and domains», *ACM Trans. Speech Lang. Process.*, vol. 3, pp. 1–35. (Cited on page 28.)

Sproat, R., Shih, C., Gale, W. and Chang, N. (1996), «A Stochastic Finite-State Word-Segmentation Algorithm for Chinese», in «Computational Linguistics», pp. 377–404. (Cited on page 1.)

Stadermann, J. and Rigoll, G. (2004), «A hybrid SVM/HMM acoustic modeling approach to automatic speech recognition», in «International Conference on Spoken Language Processing (ICSLP)», pp. 661–664. (Cited on page 308.)

Steinbiss, V., Tran, B.-H. and Ney, H. (1994), «Improvements in beam search», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 2143–2146. (Cited on page 430.)

Stephenson, T., Doss, M. and Bourlard, H. (2004), «Speech recognition with auxiliary information», *Speech and Audio Processing, IEEE Transactions on*, vol. 12 (3), pp. 189–203. (Cited on pages 75 and 139.)

Stevens, K. (2002), «Toward a model for lexical access based on acoustic landmarks and distinctive features», *The Journal of the Acoustical Society of America*, vol. 111, p. 1872. (Cited on page 51.)

Steyvers, M. and Griffiths, T. (2007), «Probabilistic topic models», *Handbook of latent semantic analysis*, vol. 427 (7), pp. 424–440. (Cited on page 259.)

Stockmeyer and Modha (2001), «Links Between Complexity Theory and Constrained Block Coding», in «SCT: Annual Conference on Structure in Complexity Theory». (Cited on page 2.)

Stolcke, A. (1995), «An efficient probabilistic context-free parsing algorithm that computes prefix probabilities», *Computational Linguistics*, vol. 21, pp. 165–201. (Cited on pages 193, 228, 229, 232, 236, 243, and 245.)

Stolcke, A. (2000a), «Entropy-based Pruning of Backoff Language Models», *Computing Research Repository (CoRR)*. (Cited on page 358.)

Stolcke, A. (2000b), «Entropy-based pruning of backoff language models», *arXiv preprint cs/0006025*. (Cited on page 372.)

Stolcke, A. (2002), «SRILM: an extensible language modeling toolkit», in «International Conference on Spoken Language Processing (ICSLP)», pp. 901–904. (Cited on pages 345, 356, 526, 528, 532, 533, 581, 592, 594, and 600.)

Stolcke, A., Bratt, H., Butzberger, J., Franco, H., Gadde, V. R., Plauché, M., Richey, C., Shriberg, E., Sönmez, K., Weng, F. *et al.* (2000), «The SRI March 2000 Hub-5 conversational speech transcription system», in «Proc. NIST Speech Transcription Workshop». (Cited on page 594.)

Strassen, V. (1969), «Gaussian elimination is not optimal», *Numerische Mathematik*, vol. 13 (4), pp. 354–356. (Cited on page 231.)

Ström, N., Hetherington, L., Hazen, T. J., Sandness, E. and Glass, J. (1999), «Acoustic modeling improvements in a segment-based speech recognizer», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 139–142. (Cited on page 51.)

Stubberud, P., Kanai, J. and Kalluri, V. (1995), «Adaptive Image Restoration of Text Images that Contain Touching or Broken Characters», in «International Conference on Document Analysis and Recognition (ICDAR)», vol. 2, pp. 778–781. (Cited on page 542.)

Sümer, Ö., Acar, U. A., Ihler, A. T. and Mettu, R. R. (2011), «Adaptive exact inference in graphical models», *The Journal of Machine Learning Research*, vol. 12, pp. 3147–3186. (Cited on page 166.)

Sundermeyer, M., Schlüter, R. and Ney, H. (2012), «LSTM Neural Networks for Language Modeling.», in «Conference of the International Speech Communication Association (Interspeech)». (Cited on page 266.)

Sung, Y.-H., Boulis, C., Manning, C. and Jurafsky, D. (2007), «Regularization, adaptation, and non-independent features improve hidden conditional random fields for phone classification», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 347 –352. (Cited on page 62.)

Sutton, C. and McCallum, A. (2006), «An introduction to conditional random fields for relational learning», *Introduction to statistical relational learning*, pp. 93–128. (Cited on page 119.)

Suzuki, K., Horiba, I. and Sugie, N. (2003), «Neural Edge Enhancer for Supervised Edge Enhancement from Noisy Images», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 25 (12), pp. 1582–1596. (Cited on page 542.)

Szöke, I., Schwarz, P., Matějka, P., Burget, L., Karafiát, M. and Černocký, J. (2005), «Phoneme based acoustics keyword spotting in informal continuous speech», in «Text, Speech and Dialogue», pp. 302–309, Springer. (Cited on page 71.)

Talbot, D. and Brants, T. (2008), «Randomized Language Models via Perfect Hash Functions», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», pp. 505–513. (Cited on page 358.)

Talbot, D. and Osborne, M. (2007), «Randomised language modelling for statistical machine translation», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», vol. 7, pp. 512–519. (Cited on page 358.)

Tappert, C., Suen, C. and Wakahara, T. (1990), «The state of the art in online handwriting recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 12 (8), pp. 787–808. (Cited on page 38.)

Tay, Y. H., Lallican, P.-M., Khalid, M., Knerr, S. and Viard-Gaudin, C. (2001), «An analytical handwritten word recognition system with word-level discriminant training», *International Conference on Document Analysis and Recognition (ICDAR)*, pp. 726–730. (Cited on page 154.)

Tebelskis, J. (1995), *Speech Recognition using Neural Networks*, Ph.D. thesis, Carnegie Mellon University. (Cited on page 308.)

Thomae, M. (2006), *Hierarchical Language Modeling for One-Stage Stochastic Interpretation of Natural Speech*, Ph.D. thesis, Technische Universität München, Munich, Germany. (Cited on pages 187 and 231.)

Thomae, M., Fabian, T., Lieb, R. and Ruske, G. (2005), «Hierarchical language models for one-stage speech interpretation», in «Conference of the International Speech Communication Association (Interspeech)», pp. 3425–3428. (Cited on page 265.)

Tomeh, N. (2012), *Discriminative Alignment Models For Statistical Machine Translation*, Ph.D. thesis, University of Paris-Sud. (Cited on page 67.)

Tomita, M. (1985), *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*, Kluwer Academic Publishers. (Cited on page 228.)

Tomita, M. (1986), «An efficient word lattice parsing algorithm for continuous speech recognition», vol. 11, pp. 1569–1572. (Cited on page 77.)

Torres, I. and Varona, A. (2001), «k-TSS language models in speech recognition systems», *Computer Speech and Language*, vol. 15 (2), pp. 127–149. (Cited on page 359.)

Toselli, A., Romero, V. and Vidal, E. (2008), «Computer Assisted Transcription of Text Images and Multimodal Interaction», in «Machine Learning for Multimodal Interaction», pp. 296–308. (Cited on page 100.)

Toselli, A., Vidal, E. and Casacuberta, F. (2011), *Multimodal Interactive Pattern Recognition and Applications*, Springer Science & Business Media. (Cited on page 83.)

Toselli, A., Romero, V., Rodriguez, L. and Vidal, E. (2007a), «Computer Assisted Transcription of Handwritten Text», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 944—-948. (Cited on page 380.)

Toselli, A. H., Romero, V. and Vidal, E. (2007b), «Viterbi based alignment between text images and their transcripts», in «Proceedings of the Workshop on Language Technology for Cultural Heritage Data», pp. 9–16. (Cited on page 102.)

Toselli, A. H., Juan, A., González, J., Salvador, I., Vidal, E., Casacuberta, F., Keysers, D. and H. Ney (2004), «Integrated Handwriting Recognition and Interpretation using Finite-State Models», *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 18 (4), pp. 519–539. (Cited on pages 564, 565, 566, 576, 578, 579, 582, 593, 598, 611, and 635.)

Toselli, A. H., Romero, V., Pastor, M. and Vidal, E. (2010), «Multimodal interactive transcription of text images», *Pattern Recognition*, vol. 43 (5), pp. 1814–1825. (Cited on pages 8, 83, and 381.)

Tóth, L. (2006), *Posterior-based speech models and their application to Hungarian speech recognition*, Ph.D. thesis, Doctoral School in Mathematics and Computer Science. University of Szeged. (Cited on page 153.)

Tranter, S. and Reynolds, D. (2006), «An overview of automatic speaker diarization systems», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 14 (5), pp. 1557 –1565. (Cited on page 87.)

Trentin, E. and Freno, A. (2009), «Unsupervised nonparametric density estimation: A neural network approach», in «International Joint Conference on Neural Networks (IJCNN)», pp. 3140–3147, IEEE. (Cited on page 308.)

Tsakalidis, S., Digalakis, V. and Neumeyer, L. (1999), «Efficient speech recognition using subvector quantization and discrete-mixture HMMs», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 2, pp. 569 –572 vol.2. (Cited on page 305.)

Turian, J., Ratinov, L. and Bengio, Y. (2010), «Word representations: a simple and general method for semi-supervised learning», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», ACL '10, pp. 384–394. (Cited on pages 254 and 255.)

Turney, P. D., Pantel, P. *et al.* (2010), «From frequency to meaning: Vector space models of semantics», *Journal of artificial intelligence research*, vol. 37 (1), pp. 141–188. (Cited on page 255.)

Tüske, Z., Sundermeyer, M., Schlüter, R. and Ney, H. (2012), «Context-Dependent MLPs for LVCSR: TANDEM, Hybrid or Both?», in «Conference of the International Speech Communication Association (Interspeech)», pp. 18–21. (Cited on page 307.)

Uchida, S., Taira, E. and Sakoe, H. (2001), «Nonuniform slant correction using dynamic programming», in «International Conference on Document Analysis and Recognition (ICDAR)», vol. 1, pp. 434–438. (Cited on page 556.)

Unger, S. (1968), «A global parser for context-free phrase structure grammars», *Communications of the ACM*, vol. 11 (4), pp. 240–247. (Cited on page 226.)

Valiant, L. G. (1975), «General context-free recognition in less than cubic time», *Journal of Computer and System Sciences*, vol. 10 (2), pp. 308–315. (Cited on page 231.)

Van Den Bosch, A. (2006), «Scalable classification-based word prediction and confusible correction», *Traitement Automatique des Langues*, vol. 46 (2), pp. 39–63. (Cited on page 247.)

Vapnik, V. N. and Vapnik, V. (1998), *Statistical learning theory*, vol. 2, Wiley New York. (Cited on page 114.)

Varga, I., Aalburg, S., Andrassy, B., Astrov, S., Bauer, J. G., Beaugeant, C., Geissler, C. and Hoge, H. (2002), «ASR in mobile phones-an industrial approach», *Speech and Audio Processing, IEEE Transactions on*, vol. 10 (8), pp. 562–569. (Cited on page 412.)

Varga, T. (2006), *Off-line Cursive Handwriting Recognition Using Synthetic Training Data*, Ph.D. thesis, University of Bern, Switzerland. (Cited on pages 568 and 576.)

Varga, T. and Bunke, H. (2003), «Generation of synthetic training data for an HMM-based handwriting recognition system», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 618–622 vol.1. (Cited on pages 576 and 635.)

Vauquois, B. and Boitet, C. (1985), «Automated translation at grenoble university», *Computational Linguistics*, vol. 11 (1), pp. 28–36. (Cited on page 30.)

Venkataramani, V. (2005), *Code breaking for automatic speech recognition*, Ph.D. thesis, Johns Hopkins University. (Cited on page 163.)

Verma, B., Gader, P. and Chen, W. (2001), «Fusion of multiple handwritten word recognition techniques», *Pattern Recognition Letters*, vol. 22 (9), pp. 991–998. (Cited on page 608.)

Vertanen, K. and MacKay, D. J. (2010), «Speech Dasher: Fast Writing using Speech and Gaze», in «CHI '10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems», pp. 595–598. (Cited on page 85.)

Veselỳ, K., Ghoshal, A., Burget, L. and Povey, D. (2013), «Sequence-discriminative training of deep neural networks.», in «Conference of the International Speech Communication Association (Interspeech)», pp. 2345–2349. (Cited on pages 613 and 635.)

Viard-Gaudin, C., Lallican, P., Knerr, S. and Binter, P. (1999), «The IRESTE On/Off (IRONOFF) dual handwriting database», in «International Conference on Document Analysis and Recognition (ICDAR)», pp. 455 –458. (Cited on page 646.)

Vidal, E. (1997), «Finite-state speech-to-speech translation», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 111–114. (Cited on page 32.)

Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F. and Carrasco, R. (2005a), «Probabilistic finite-state machines - part I», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 27 (7), pp. 1013–1025. (Cited on page 169.)

Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F. and Carrasco, R. (2005b), «Probabilistic finite-state machines - part II», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 27 (7), pp. 1026–1039. (Cited on pages 169, 265, and 345.)

Vilar, J. M. (2000), «Improve the learning of subsequential transducers by using alignments and dictionaries», in «Grammatical Inference: Algorithms and Applications», pp. 298–311, Springer. (Cited on page 266.)

Vilar, J. M. (2008), «Efficient computation of confidence intervals for word error rates», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 5101–5104. (Cited on page 97.)

Vilar, J. M., Castro-Bleda, M. J., Zamora-Martínez, F., España-Boquera, S., Gordo, A., Llorens, D., Marzal, A., Prat, F. and Gorbe-Moya, J. (2009), «State: A Flexible System for Document Processing and Text Transcription», in «Conferencia de la Asociación Española para la Inteligencia Artificial», pp. 467–476. (Cited on page 641.)

Vilar, J. M., Castro-Bleda, M. J., Zamora-Martínez, F., España-Boquera, S., Gordo, A., Llorens, D., Marzal, A., Prat, F. and Gorbe-Moya, J. (2010), «A Flexible System for Document Processing and Text Transcription», in «Current Topics in Artificial Intelligence», vol. 5988 of *LNCS*, pp. 291–300. (Cited on pages 83 and 640.)

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.-A. (2010), «Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion», *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408. (Cited on page 264.)

Vinciarelli, A. (2002), «A survey on off-line cursive word recognition», *Pattern Recognition*, vol. 35 (7), pp. 1433–1446. (Cited on page 579.)

Vinciarelli, A. (2003), *Offline Cursive Handwriting: From Word To Text Recognition*, Ph.D. thesis, University of Bern, Switzerland. (Cited on page 565.)

Vinciarelli, A., Bengio, S. and Bunke, H. (2004), «Offline Recognition of Unconstrained Handwritten Texts Using HMMs and Statistical Language Models», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 26 (6), pp. 709–720. (Cited on page 579.)

Vinciarelli, A. and Luettin, J. (2001), «A new normalization technique for cursive handwritten words», *Pattern Recognition Letters*, vol. 22 (9), pp. 1043–1050. (Cited on pages 544, 552, and 555.)

Vinciarelli, A. and Lüttin, J. (2000), «Off-line cursive script recognition based on continuous density HMM», in Schomaker, L. R. B. and Vuurpijl, L. G., eds., «International Workshop on Frontiers in Handwriting Recognition (IWFHR)», pp. 493–498. (Cited on page 3.)

Vintsyuk, T. K. (1971), «Element-wise recognition of continuous speech composed of words from a specified dictionary», *Cybernetics and Systems Analysis*, vol. 7, pp. 361–372. (Cited on page 7.)

Voigtlaender, P., Doetsch, P., Wiesler, S., Schlüter, R. and Ney, H. (2015), «Sequence-Discriminative Training of Recurrent Neural Networks», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 2100–2104. (Cited on pages 612 and 636.)

von Ahn, L. and Dabbish, L. (2004), «Labeling images with a computer game», in «Proceedings of the SIGCHI conference on Human factors in computing systems», CHI '04, pp. 319–326. (Cited on page 82.)

von Ahn, L., Maurer, B., McMillen, C., Abraham, D. and Blum, M. (2008), «recaptcha: Human-based character recognition via web security measures», *Science*, vol. 321 (5895), pp. 1465–1468. (Cited on page 82.)

Wang, R., Utiyama, M., Goto, I., Sumita, E., Zhao, H. and Lu, B.-L. (2013), «Converting Continuous-Space Language Models into N-gram Language Models for Statistical Machine Translation»,

in «Proc. of Empirical Methods in Natural Language Processing (EMNLP)», pp. 845–850. (Cited on pages 261 and 369.)

Wang, S.-J. and Jin, C.-Z. (2006), «An extension of Earley's algorithm for extended grammars», in LIU, G., TAN, V. and HAN, X., eds., «Computational Methods», pp. 1147–1152. (Cited on page 229.)

Ward, D. J., Blackwell, A. F. and MacKay, D. J. C. (2000), «Dasher — a data entry interface using continuous gestures and language models», in «Proceedings of the 13th annual ACM symposium on User interface software and technology», UIST '00, pp. 129–137. (Cited on page 36.)

Ward, N. G., Vega, A. and Baumann, T. (2012), «Prosodic and temporal features for language modeling for dialog», *Speech Communication*, vol. 54 (2), pp. 161–174. (Cited on pages 55 and 259.)

Watanabe, T., Tsukada, H. and Isozaki, H. (2009), «A succinct N-gram language model», in «ACL-IJCNLP '09: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers», pp. 341–344. (Cited on page 358.)

Weber, K., Bengio, S. and Bourlard, H. (2001), «Speech recognition using advanced HMM2 features», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 65–68. (Cited on page 309.)

Wegmann, S. and Gillick, L. (2010), «Why has (reasonably accurate) Automatic Speech Recognition been so hard to achieve?», *ArXiv e-prints*. (Cited on page 290.)

Weimann, O. (1997), *Accelerating dynamic programming*, Ph.D. thesis, Massachusetts Institute of Technology. (Cited on page 436.)

Weinstein, E. (2009), *Search Problems for Speech and Audio Sequences*, Ph.D. thesis, Courant Institute of Mathematical Sciences. (Cited on page 59.)

Weischedel, R., Schwartz, R., Palmucci, J., Meteer, M. and Ramshaw, L. (1993), «Coping with ambiguity and unknown words through probabilistic models», *Computational Linguistics*, vol. 19 (2), pp. 361–382. (Cited on page 29.)

Wellekens, C. (1987), «Explicit time correlation in hidden Markov models for speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 12, pp. 384–386. (Cited on page 290.)

Weng, F., Stolcke, A. and Sankar, A. (1998), «Efficient Lattice Representation and Generation», in «International Conference on Spoken Language Processing (ICSLP)», vol. 6, pp. 2531–2534. (Cited on page 493.)

Wessel, F. (2002), *Word Posterior Probabilities for Large Vocabulary Continuous Speech Recognition*, Ph.D. thesis, RWTH Aachen University. (Cited on page 491.)

Wessel, F., Schluter, R. and Ney, H. (2001), «Explicit word error minimization using word hypothesis posterior probabilities», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 33–36. (Cited on page 97.)

Whittaker, E. W. D., Van Thong, J.-M. and Moreno, P. (2001), «Vocabulary independent speech recognition using particles», in «Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on», pp. 315–318. (Cited on page 251.)

Wienecke, M., Fink, G. A. and Sagerer, G. (2002), «Experiments in unconstrained offline handwritten text recognition», in «International Workshop on Frontiers in Handwriting Recognition (IWFHR)». (Cited on pages 270 and 599.)

Wilson, A. and Bobick, A. (1999), «Parametric hidden Markov models for gesture recognition», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 21 (9), pp. 884 –900. (Cited on page 139.)

Winter, J., Bonsangue, M. M. and Rutten, J. (2011), «Context-free languages, coalgebraically», in «Proceedings of the 4th international conference on Algebra and coalgebra in computer science», CALCO'11, pp. 359–376. (Cited on page 179.)

Wirth, N. (1977), «What can we do about the unnecessary diversity of notation for syntactic definitions?», *Communications of the ACM*, vol. 20, pp. 822–823. (Cited on page 189.)

Wolf, C. (2010), «Document ink bleed-through removal with two hidden Markov random fields and a single observation field», *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 32 (3), pp. 431–447. (Cited on page 542.)

Wong, K. Y., Casey, R. G. and Wahl., F. M. (1982), «Document Analysis system», *IBM Journal of Research and Developement*, vol. 26 (6), pp. 647–655. (Cited on page 544.)

Woods, W. A. (1970), «Transition network grammars for natural language analysis», *Communications of the ACM*, vol. 13 (10), pp. 591–606. (Cited on pages 179, 187, 188, and 353.)

Woods, W. A. (2010), «The right tools: Reflections on computation and language», *Computational Linguistics*, vol. 36 (4), pp. 601–630. (Cited on page 189.)

Woszczyna, M. (1998), *Fast Speaker Independent Large Vocabulary Continuous Speech Recognition*, Ph.D. thesis, Karlsruhe Institute of Technology. (Cited on pages 151 and 306.)

Wu, S.-L., Kingsbury, E., Morgan, N. and Greenberg, S. (1998), «Incorporating information from syllable-length time scales into automatic speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 2, pp. 721 –724 vol.2. (Cited on page 45.)

Xu, P. and Jelinek, F. (2004), «Random forests in language modeling», in «Proc. of Empirical Methods in Natural Language Processing (EMNLP)». (Cited on page 267.)

Xue, J. and Zhao, Y. (2005), «Improved Confusion Network Algorithm and Shortest Path Search from Word Lattice», vol. 1, pp. 853–856. (Cited on pages 76 and 78.)

Yamamoto, H., Isogai, S. and Sagisaka, Y. (2001), «Multi-Class Composite N-gram language model for spoken language processing using multiple word clusters», in «Proceedings of the annual meeting on Association for Computational Linguistics (ACL)», ACL '01, pp. 531–538. (Cited on page 253.)

Ye, X., Cheriet, M. and Suen, C. Y. (2001), «A generic method of cleaning and enhancing handwritten data from business forms», *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 4, pp. 84–96. (Cited on page 542.)

Young, S. (2007), «ATK: An application toolkit for HTK», Tech. rep., Machine Intelligence Laboratory, Cambridge University Engineering Department, Trumpington Street, Cambridge, CB2 1PZ, England. (Cited on page 325.)

Young, S., Russell, N. and Thornton, J. (1989), «Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems», Tech. Rep. CUED/F–INFENG/TR38, Cambridge University Engineering Dept. (Cited on pages 7, 387, and 395.)

Young, S. J., Woodland, P. C. and Byrne, W. J. (1993), «HTK: Hidden Markov Model Toolkit V1.5», Tech. rep., Cambridge University Engineering Department Speech Group and Entropic Research Laboratories Inc. (Cited on pages 323, 582, and 629.)

Younger, D. H. (1967), «Recognition and parsing of context-free languages in time n 3», *Information and control*, vol. 10 (2), pp. 189–208. (Cited on page 227.)

Yu, D., Deng, L., Gong, Y. and Acero, A. (2009), «A Novel Framework and Training Algorithm for Variable-Parameter Hidden Markov Models», *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 17 (7), pp. 1348–1360. (Cited on page 140.)

Yu, D., Eversole, A., Seltzer, M., Yao, K., Huang, Z., Guenter, B., Kuchaiev, O., Zhang, Y., Seide, F., Wang, H. *et al.* (2014), «An introduction to computational networks and the computational network toolkit», Tech. rep., Tech. Rep. MSR, Microsoft Research, 2014, http://codebox/cntk. (Cited on page 433.)

Yu, G. T. (2008), *Efficient Error Correction for Speech Systems using Constrained Re-recognition*, Master's thesis, MIT Department of Electrical Engineering and Computer Science. (Cited on page 48.)

Yu, S.-Z. (2010), «Hidden semi-Markov models», *Artificial Intelligence*, vol. 174 (2), pp. 215–243, special Review Issue. (Cited on page 295.)

Zamora, F., España, S., Gorbe, J. and Castro, M. J. (2006), «Entrenamiento de Modelos de Lenguaje Conexionistas con Grandes Vocabularios», in «IV Jornadas en Tecnologia del Habla», pp. 141–144. (Cited on page 639.)

Zamora-Martínez, F. (2012), *Aportaciones al Modelado Conexionista de Lenguaje y su aplicación al Reconocimiento Automático de Secuencias y Traducción Estadística*, Ph.d. thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, advisor: Castro-Bleda, M.J. (Cited on pages 263, 264, 329, 369, 370, 373, 383, 503, 592, and 599.)

Zamora-Martínez, F., Castro-Bleda, M. and España-Boquera, S. (2009a), «Fast Evaluation of Connectionist Language Models», in «International Work-Conference on Artificial Neural Networks», vol. 5517 of *Lecture Notes in Computer Science*, pp. 33–40, Springer. (Cited on pages 265, 369, 370, 373, 377, 383, 531, 532, and 638.)

Zamora-Martínez, F., Castro-Bleda, M. J. and Schwenk, H. (2010), «N-gram-based Machine Translation enhanced with Neural Networks for the French-English BTEC-IWSLT'10 task», in «Proc. of the International Workshop on Spoken Language Translation (IWSLT)», pp. 45–52. (Cited on page 534.)

Zamora-Martínez, F., España-Boquera, S. and Castro-Bleda, M. J. (2007), «Behaviour-based clustering of neural networks applied to document enhancement», in «IWANN'07: Proceedings of the 9th international work conference on Artificial neural networks», pp. 144–151. (Cited on pages 544 and 640.)

Zamora-Martínez, F., Castro-Bleda, M., Tortajada-Velert, S. and España-Boquera, S. (2009b), «A Connectionist approach to Part-Of-Speech Tagging», in «International Conference on Neural Computation», pp. 421–426. (Cited on page 641.)

Zamora-Martínez, F., Castro-Bleda, M., Tortajada-Velert, S. and España-Boquera, S. (2009c), «Adding morphological information to a connectionist Part-Of-Speech tagger», in «Current Topics in Artificial Intelligence», vol. 5988 of *Lecture Notes in Computer Science*, pp. 191–200. (Cited on page 641.)

Zamora-Martínez, F., Castro-Bleda, M. J., España-Boquera, S. and Gorbe-Moya, J. (2009), «Mejoras del reconocimiento de palabras manuscritas aisladas mediante un clasificador específico para palabras cortas», in «Conferencia de la Asociación Española para la Inteligencia Artificial», pp. 539–548. (Cited on page 639.)

Zamora-Martínez, F., Castro-Bleda, M. J., España-Boquera, S. and Gorbe, J. (2010), «Improving isolated handwritten word recognition using a specialized classifier for short words», in «Proceedings of the Current topics in artificial intelligence, and 13th conference on Spanish association for artificial intelligence», CAEPIA'09, pp. 61–70. (Cited on pages 575 and 637.)

Zamora-Martinez, F., España-Boquera, S., Castro-Bleda, M. J. and De-Mori, R. (2012), «Cache Neural Network Language Models Based on Long-Distance Dependencies for a Spoken Dialog System», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», pp. 4993–4996. (Cited on pages 12, 260, 517, 577, 634, and 640.)

Zamora-Martínez, F., España-Boquera, S., Castro-Bleda, M. and Palacios-Corella, A. (2015), «Fall Back Skipping NNLM: Efficient Neural Network Language Models by Precomputation and Stochastic Training», *Speech Communication (submitted to)*, pp. –. (Cited on page 636.)

Zamora-Martínez, F., Castro-Bleda, M., España-Boquera, S. and Gorbe, J. (2010a), «Improving Isolated Handwritten Word Recognition Using a Specialized Classifier for Short Words», in «Current Topics in Artificial Intelligence», vol. 5988 of *Lecture Notes in Computer Science*, pp. 61–70. (Cited on pages 578, 604, 605, 607, 608, 609, and 638.)

Zamora-Martínez, F., Castro-Bleda, M., España-Boquera, S. and Gorbe-Moya, J. (2010b), «Unconstrained Offline Handwriting Recognition using Connectionist Character N-grams», in «International Joint Conference on Neural Networks (IJCNN)», pp. 4136–4142. (Cited on pages 376, 578, 599, 600, 601, 602, 603, and 637.)

Zamora-Martínez, F., Frinken, V., España-Boquera, S., Castro-Bleda, M., Fischer, A. and Bunke, H. (2014), «Neural network language models for off-line handwriting recognition», *Pattern Recognition*, vol. 47 (4), pp. 1642–1652. (Cited on pages 526, 531, 578, 592, 593, 594, 595, 596, 597, 600, 612, and 638.)

Zen, H., Tokuda, K. and Black, A. W. (2009), «Review: Statistical parametric speech synthesis», *Speech Communication*, vol. 51, pp. 1039–1064. (Cited on pages 41, 137, and 285.)

Zen, H., Tokuda, K. and Kitamura, T. (2007), «Reformulating the HMM as a trajectory model by imposing explicit relationships between static and dynamic feature vector sequences», *Computer Speech and Language*, vol. 21 (1), pp. 153–173. (Cited on pages 136, 290, 300, and 302.)

Zhang, X. and Zhao, Y. (2002), «Minimum perfect hashing for fast N-gram language model lookup», in «International Conference on Spoken Language Processing (ICSLP)», pp. 16–20. (Cited on page 357.)

Zhou, J., Tian, Y., Shi, Y., Huang, C. and Chang, E. (2004), «Tone articulation modeling for Mandarin spontaneous speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 1, pp. 997–1000 vol.1. (Cited on page 48.)

Zhu, Q. and Alwan, A. (2000), «On the use of variable frame rate analysis in speech recognition», in «Acoustics, Speech, and Signal Processing, IEEE International Conference on (ICASSP)», vol. 3, pp. 1783–1786. (Cited on pages 422 and 423.)

Zimmermann, M. and Bunke, H. (2002a), «Automatic segmentation of the IAM off-line database for handwritten English text», in «International Conference on Pattern Recognition (ICPR)», vol. 4, pp. 35–39. (Cited on page 567.)

Zimmermann, M. and Bunke, H. (2002b), «Hidden Markov model length optimization for handwriting recognition systems», in «Frontiers in Handwriting Recognition (ICFHR), International Conference on», pp. 369–374. (Cited on pages 299 and 582.)

Zinger, S., Nerbonne, J. and Schomaker, L. (2009), «Text-image alignment for historical handwritten documents», in «IS&T/SPIE Electronic Imaging», pp. 724 703–724 703, International Society for Optics and Photonics. (Cited on page 68.)

Zweig, G. (1998), *Speech recognition with dynamic Bayesian networks*, Ph.D. thesis, PhD thesis, University of California, Berkley, USA. (Cited on pages 146 and 147.)

Zweig, G. (2009), «New methods for the analysis of repeated utterances», in «Conference of the International Speech Communication Association (Interspeech)», pp. 2791–2794. (Cited on page 80.)