UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Aportaciones al modelado conexionista de lenguaje y su aplicación al reconocimiento de secuencias y traducción automática

Tesis Doctoral

Noviembre del 2012

Francisco Zamora Martínez
Directora: Dra. María José Castro Bleda

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Aportaciones al modelado conexionista de lenguaje y su aplicación al reconocimiento de secuencias y traducción automática

Tesis Doctoral
realizada por Francisco Zamora Martínez
para optar al grado de Doctor en Informática
dirigida por la Dra. María José Castro Bleda

Noviembre del 2012

Tesis financiada parcialmente por el Gobierno de España en los proyectos de investigación TIN2005-08660-C04-02 (EDECAN), TIN2006-12767 (STATE), TIN2010-18958 (HITITA), y por la Generalitat Valenciana a través del proyecto de investigación GV06/302 (MORE).

A Josepa por apoyarme en estos últimos años de escritura. A mis padres, Natividad y Crescencio, y hermanos, José y Nati, por creer en mí.

AGRADECIMIENTOS

Quiero aprovechar este espacio para agradecer el apoyo, críticas, y buenas intenciones que he encontrado en la gente en el camino de obtener el título de Doctor. Doy las gracias a mis amigos de Aldaia, que he tenido muy olvidados en estos últimos años, pero que siguen confiando en mí. A Germán, Jorge y Jesús compañeros de andadura durante la carrera, y compañeros en el doctorado, gracias por las discusiones, comentarios, curiosidades y demás cosas que he podido aprender con vosotros. A Lucena, Ximo, Carlos, Julien, Fidel, Antonio, Pablo por los años de música y conciertos. A Noelia por acompañarme durante un tiempo en la vida solapado con esta tesis. A Salva por estos años haciendo guía y ayudando en todo lo que ha podido siempre, aportando un punto de vista original, implicados ambos en la programación del software aquí utilizado. A María José por la dirección desde mi proyecto final de carrera, hasta el final con las sucesivas correcciones que esta tesis ha necesitado para ser lo que es. A Jon, Emilio y Encarna por su apoyo en diferentes ocasiones para encontrar trabajo. A la gente que ha pasado por el laboratorio L302, Sergio, David, Joan, Marcos, Lucía, Ximo, por su amistad. A la gente del departamento de Ciencias Físicas, Matemáticas y de la Computación de la Universidad CEU Cardenal Herrera, donde actualmente estoy trabajando, por haberme dado la oportunidad de poder poner un pie en el mundo de la enseñanza universitaria, a Javi, Nuria Rosillo, Juan, Pablo, Rafa Pérez, Rafa Ramírez, Nuria Llull, Manolo, Luís, Victor, Jordi, Antonio, Fernando, que juntos nos esforzamos por dar lo mejor en clase, y que me han ofrecido toda su confianza para que termine esta tesis. A los alumnos Fran, Joan, Vicent y Jose Manuel por implicarse en la competición del SWERC, que tanto disfruto. A la gente del DSIC, así como a gente del ITI, que he ido conociendo estos años. A Renato de Mori por su valiosa aportación a este trabajo. A la gente del LIUM en Le Mans, Patrik, Loïc, Kashif, Benoit, por ofrecerme su amistad durante mi estancia veraniega del 2010 en su universidad, y por supuesto a Holger Schwenk por acogerme y apoyar mis ideas. A Maxim por introducirme en el mundo de la traducción automática, y Josep María por recibirme en aquel verano del 2010. Al antiguo grupo IAM de la Universidad de Bern, Volkmar Frinken, Andreas Fischer y Horst Bunke, por los trabajos colaborativos que han dado una mayor profundidad a los resultados presentados en esta tesis. A todos aquellos de los que puede que me olvide que han repercutido de alguna forma en este trabajo, y que me han apoyado para llegar hasta aquí. A todos vosotros, ¡Gracias!

RESUMEN

El procesamiento del lenguaje natural es un área de la inteligencia artificial, en particular, del reconocimiento de formas. Es un campo multidisciplinar que estudia el lenguaje humano, tanto oral como escrito. Se ocupa de la formulación e investigación de mecanismos computacionales para la comunicación entre personas y ordenadores, haciendo uso de lenguajes naturales. Es un área de investigación en continuo desarrollo, y este trabajo se centra únicamente en la parte relacionada con el modelado de lenguaje, y su aplicación a diversas tareas: reconocimiento/comprensión de secuencias y traducción automática estadística.

Concretamente, esta tesis tiene su foco central en los denominados modelos conexionistas de lenguaje, esto es, modelos de lenguaje basados en redes neuronales. Los excelentes resultados de estos modelos en diversas áreas del procesamiento del lenguaje natural han motivado el desarrollo de este estudio.

Debido a determinados problemas computacionales que adolecen los modelos conexionistas de lenguaje, los sistemas que aparecen en la literatura se construyen en dos etapas totalmente desacopladas. En la primera fase se encuentra, a través de un modelo de lenguaje estándar, un conjunto de hipótesis factibles, asumiendo que dicho conjunto es representativo del espacio de búsqueda en el cual se encuentra la mejor hipótesis. En segundo lugar, sobre dicho conjunto, se aplica el modelo conexionista de lenguaje y se extrae la hipótesis con mejor puntuación. A este procedimiento se le denomina repuntuación de las mejores soluciones o "rescoring" si usamos el término en inglés.

Este escenario motiva los objetivos científicos principales de esta tesis:

- Proponer técnicas para reducir drásticamente el coste computacional de los modelos conexionistas de lenguaje degradando lo mínimo posible la calidad de la solución encontrada.

- Estudiar el efecto que tiene la integración de los modelos conexionistas de lenguaje en el proceso de búsqueda de las tareas propuestas.

- Proponer modificaciones y variaciones del modelo original que permitan mejorar su calidad y adaptación al dominio.

- Aplicar los modelos conexionistas de lenguaje a tareas de reconocimiento de secuencias y traducción automática.

Todos los algoritmos necesarios para el desarrollo de esta tesis han sido implementados en C++ y el lenguaje de "scripting" Lua. Asimismo, se han comparado estas implementaciones con aquellas que son estándar en las diferentes tareas abordadas. En cuanto a los resultados, la incorporación de modelos conexionistas de lenguaje en dichas tareas ha logrado mejorar ampliamente los sistemas de referencia desarrollados:

- resultados competitivos en una tarea de reconocimiento y comprensión automática del habla;
- mejora del estado del arte en reconocimiento de escritura manuscrita;
- resultados al nivel del estado del arte en traducción automática estadística, como se demuestra en el posicionamiento de los sistemas presentados a evaluaciones internacionales.

La integración de estos modelos en el proceso de búsqueda para tareas de reconocimiento de secuencias ha logrado ser competitivo a nivel de coste computacional. Sin embargo, la integración en tareas de traducción automática precisa de un desarrollo más profundo, pues a nivel computacional el coste del sistema sigue siendo algo elevado.

RESUM

El processament del llenguatge natural és una àrea de la intel·ligència artificial, particularment, del reconeixement de formes. És un camp multi-disciplinar que estudia el llenguatge humà, tant oral com escrit. S'ocupa de la formulació i investigació de mecanismes computacionals per a la comunicació entre persones i ordinadors, mitjançant llenguatges naturals. És una àrea d'investigació en continu desenvolupament, i aquest treball és focalitza únicament en la part relacionada amb el modelat del llenguatge, i la seua aplicació a diverses tasques: reconeixement/comprensió de seqüències i traducció automàtica estadística.

Concretament, aquesta tesis té el seu fil conductor en els anomenats models conexionistes de llenguatge, es a dir, models de llenguatge basats en xarxes neuronals. Els bons resultats d'aquestos models en diverses àrees del processament del llenguatge natural ha motivat el desenvolupament d'aquest estudi.

Degut a determinats problemes computacionals que patixen els models conexionistes de llenguatge, els sistemes que poden trobar-se a la literatura és construeixen en dues etapes totalment desacoblades. En la primera fase es genera, mitjançant un model estàndard de llenguatge, un conjunt d'hipòtesis factibles, assumint que l'anomenat conjunt és representatiu de l'espai de recerca on pot trobar-se la millor de les hipòtesis. En segon lloc, s'utilitza el model conexionista de llenguatge i és cerca la hipòtesis amb millor puntuació dintre de l'anterior conjunt. Aquest procediment s'anomena repuntuació de les millors hipòtesis o "rescoring" si fem ús del terme anglès.

Aquest escenari motiva els objectius científics principals d'aquesta tesis:

- Proposar tècniques per a reduir dràsticament el cost computacional dels models conexionistes de llenguatge perdent el mínim possible de qualitat en la solució trobada.
- Estudiar l'efecte que té la integració dels models conexionistes de llenguatge en el procés de cerca a les tasques proposades.
- Proposar modificacions i variacions del model original que puguen permetre millorar la qualitat dels resultats i l'adaptació al domini.
- Utilitzar els models conexionistes de llenguatge en tasques de reconeixement de seqüències i traducció automàtica.

Tots els algorismes necessaris per al desenvolupament d'aquesta tesis han sigut implementats en C++ i el llenguatge de "scripting" Lua. Tanmateix, han sigut comparades aquestes implementacions amb altres reconegudes com estàndard a les tasques abordades. Pel que fa als resultats, la incorporació dels models conexionistes de llenguatge en les citades tasques ha assolit millores àmplies respecte al sistema de referència desenvolupat:

- resultats competitius en una tasca de reconeixement i comprensió automàtica de la parla;
- millora de l'estat de l'art en reconeixement d'escriptura manuscrita;
- resultats al nivell de l'estat de l'art en traducció automàtica estadística, com mostra el posicionament dels sistemes presentats a avaluacions internacionals.

La integració d'aquestos models en el procés de cerca per a tasques de reconeixement de seqüències ha sigut competitiu a nivell de cost computacional. Malgrat això, la integració en tasques de traducció automàtica necessita un desenvolupament més profund, ja que a nivell computacional el cost del sistema continua sent una mica elevat.

ABSTRACT

Natural Language Processing is an area of Artificial Intelligence, in particular, of Pattern Recognition. It is a multidisciplinary field that studies human language, both oral and written. It deals with the development and research of computational mechanisms for communication between people and computers, using natural languages. It is a research area constantly evolving, and this work focuses only on the part related to language modeling, and its application to various tasks: recognition/understanding of sequences and statistical machine translation.

Specifically, this thesis focus its interest on the so-called connectionist language models (or continuous space language models), i.e., language models based on neural networks. Their excellent performance in various Natural Language Processing areas has motivated this study.

Because of certain computational problems suffered by connectionist language models, the most widespread approach followed by the systems that currently use these models in literature, is based on two totally decoupled stages. At a first stage, using a standard and cheaper language model, a set of feasible hypotheses, assuming that this set is representative of the search space in which the best hypothesis is located, is generated. In a second stage, on this set, a connectionist language model is applied and a rescoring of the list of hypotheses is done.

This scenario motivates the main scientific goals of this thesis:

- Developing techniques in order to dramatically reduce the computational cost of connectionist language models degrading, as less as possible, the quality of the final solution.
- Studying the effect of integrating the connectionist language models into the search process of the proposed tasks.
- Developing some variations and extensions of the original model in order to improve its quality and to fulfill its context domain adaptation.
- Applying neural network language models to sequence recognition and machine translation tasks.

All algorithms were implemented in C++ and Lua programming languages. Also, these implementations have been compared with those that are considered standard in each task. Neural network language models achieve very interesting improvements of quality over the reference baseline systems:

- competitive results are achieved in an automatic speech recognition and spoken language understanding task;
- improving the state-of-the-art in handwritten text recognition;
- state-of-the-art results in statistical machine translation, as stated in positioning systems presented to international evaluation campaigns.

Finally, for sequence recognition tasks, the integration of neural network language models into the decoding stage achieves very competitive computational costs. However, their integration in machine translation tasks requires a deeper development because the computation cost of the system is still somewhat high.

PREFACIO

El autor

Esta tesis es mi pequeña contribución al mundo de la investigación en el campo de la inteligencia artificial y el reconocimiento de formas. Mi granito de arena es aportar nuevas ideas sobre el uso de modelos de lenguaje basados en redes neuronales (NN LMs). Estos modelos despiertan en estos momentos un gran interés en la comunidad, pese a que durante mucho tiempo han estado olvidados.

Los términos "modelos conexionistas de lenguaje", "modelos de lenguaje basados en redes neuronales", "modelos de lenguaje en el espacio continuo", que aparecen en este trabajo, así como sus correspondientes en inglés, "Connectionist Language Models", "Neural Network Language Models" y "Continuous Space Language Models", que encontramos en la literatura, hacen referencia exactamente al mismo concepto, y por tanto son sinónimos. Sin ánimo de enredar al lector, haremos uso de unos u otros en función del texto que se esté escribiendo, evitando repeticiones que puedan dificultar la lectura. Con la intención de unificar la terminología usaremos el acrónimo NN LM o su plural NN LMs para referirnos a cualquiera de estos términos.

La escritura de este trabajo se ha dividido por partes, tratando que sean lo más autocontenidas posible:

Introducción: Se exponen las razones de este trabajo así como las formulaciones y algoritmos comunes a todas las partes.

Parte I **Aportaciones al modelado conexionista de lenguaje**: Esta parte se ha dividido en cuatro capítulos. El primero especifica formalmente los modelos estadísticos de lenguaje y los modelos conexionistas de lenguaje. El segundo se centra en detallar las aportaciones más importantes de esta tesis a dicho campo de investigación. El tercero detalla una de las aportaciones más importantes de este texto, la evaluación rápida de los modelos conexionistas de lenguaje. Finalmente el cuarto capítulo sintetiza otra importante aportación, la extensión de los NN LMs a modelos de lenguaje tipo caché.

Parte II **Aplicación de NN LMs al reconocimiento de secuencias**: Esta parte se ha dividido en tres capítulos. El primero de ellos formaliza la aproximación basada en modelos ocultos de Markov hibridados con redes neuronales para el reconocimiento automático de secuencias. El segundo describe los resultados que se han obtenido al aplicar los modelos conexionistas de lenguaje a una tarea de comprensión del lenguaje hablado. El tercero muestra y discute los experimentos realizados en tareas de reconocimiento automático de escritura manuscrita con modelos conexionistas de lenguaje.

Parte III **Aplicación de NN LMs a la traducción automática estadística**: Esta parte vuelve a tener tres capítulos, si bien la estructura es diferente. El primero de ellos formaliza y explica el problema de la traducción automática estadística. El segundo detalla el sistema de traducción desarrollado para esta tesis. Dicho sistema está basado en el algoritmo de dos pasos e incorpora el modelo conexionista de lenguaje de forma totalmente integrada. El tercer capítulo expone los resultados que se han obtenido al utilizar modelos conexionistas de lenguaje con este sistema de traducción. También detalla experimentos realizados con modelos conexionistas de lenguaje y herramientas estándar que están disponibles en internet de forma libre.

Parte IV **Conclusiones**: Resume los puntos más importantes de todo el trabajo y al mismo tiempo recapitula los objetivos que se han logrado. También expone los trabajos futuros que quedan pendientes y que pretendemos llevar a cabo como continuación de la investigación iniciada.

Apéndices: Partes del trabajo que no son relevantes para entender las aportaciones y desarrollo de la tesis. No obstante, son un valor añadido con el que acompañar el resto del texto. Aquí también se incluye un amplio resumen en inglés de todo el trabajo. Muchas partes de este resumen hacen referencia a las publicaciones que este trabajo ha dado lugar.

La introducción y la primera parte son dependencia del resto de la tesis. Por esta razón se recomienda su lectura. La introducción cuenta mucha de la metodología utilizada en la tesis, siendo la mayor parte de esta una metodología estándar. La primera parte es la más importante, pues introduce la tesis en el marco del modelado de lenguaje. Un lector conocedor del campo puede saltarse el primero de los capítulos de esta parte. Los capítulos 3 y 4 son de lectura obligada para entender las aportaciones principales de este texto.

El resto de partes pueden ser leídas en el orden que el lector prefiera, en función de la tarea que más interés le suscite. Dentro de cada una de ellas debe mantenerse el orden de lectura de los capítulos, si bien un lector conocedor del marco formal de cada una de las tareas puede saltarse el primer capítulo de cada parte.

Las figuras y tablas de este trabajo están en inglés para que puedan ser enlazadas y entendidas por el lector internacional. Desde el apéndice D, que es un resumen en inglés de las aportaciones más importantes, se hace referencia a estas figuras y tablas. La leyenda de las

mismas se ha traducido al inglés para facilitar la lectura del resumen. Del mismo modo, los acrónimos utilizados en todo el texto son de origen inglés, siendo redactados directamente en inglés dicho glosario y el capítulo de conclusiones. Se han tomado también algunas libertades a la hora de representar valores numéricos[1], así como en determinadas expresiones que han sido tomadas directamente del inglés.

La metodología seguida en esta tesis ha dado como fruto el desarrollo de parte de una herramienta, denominada `April` ("A Pattern Recognizer In Lua"), que incorpora todos los algoritmos aquí descritos. Esta herramienta es resultado de la investigación conjunta realizada dentro del grupo de investigación, siendo muchos de los algoritmos implementados parte también de la tesis de Salvador España Boquera [España-Boquera, 2012].

[1] Por ejemplo, no utilizar puntuación para separar miles, millones, etc., pero sí dejar un pequeño espacio entre los números para mejorar la legibilidad

ÍNDICE GENERAL

ÍNDICE DE TABLAS

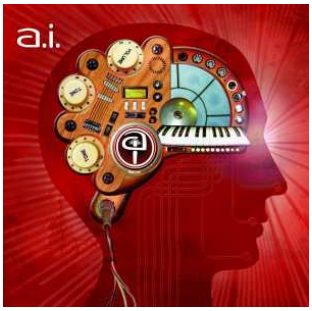
ÍNDICE DE FIGURAS

ÍNDICE DE ALGORITMOS

Introducción

1

INTRODUCCIÓN



Índice

Este capítulo introductorio establece un nexo común a todas las tareas abordadas en esta tesis. Define las ecuaciones básicas que estarán presentes en todos los apartados, así como las medidas de evaluación y los algoritmos comunes más importantes.

1.1 Motivación

El campo del reconocimiento de formas abarca una multitud de tareas diferentes, y una variedad igual o más grande de técnicas para resolver los problemas asociados a dichas tareas. Esta tesis se centra en algunos de los problemas de la decodificación en sistemas automáticos de reconocimiento de secuencias (voz y escritura) y traducción de texto, siendo el foco concreto del trabajo el área de la lingüística computacional. En ella, el procesamiento del lenguaje natural tiene una gran relevancia, ya que persigue uno de los objetivos más ambiciosos de la inteligencia artificial: lograr que la comunicación con las máquinas sea en un lenguaje lo más natural y *humano* posible. Actualmente son las técnicas estadísticas las que mejores resultados obtienen. Esto es debido, en gran medida, al incremento de la capacidad computacional de los ordenadores, de los recursos tecnológicos que tenemos a nuestra disposición y de la gran cantidad de información digital existente en internet.

En lo más básico, la aproximación estadística al reconocimiento de formas se construye sobre dos modelos que sirven para medir:

- la verosimilitud de la muestra dado el modelo y una hipótesis del sistema,
- y la probabilidad a priori de dicha hipótesis.

En su aplicación al reconocimiento automático del habla/escritura, o a la traducción automática, la probabilidad a priori de la respuesta del sistema se obtiene con lo que se conoce como modelo de lenguaje. Dicho modelo permite asignar a una hipótesis cual es la probabilidad de que pertenezca al conjunto de frases *posibles* de un determinado lenguaje natural.

En las áreas del procesamiento del lenguaje natural, y la lingüística computacional en general, no se ha prestado nunca demasiada atención a la posibilidad de usar modelos de lenguaje basados en redes neuronales (NN LMs) con el fin de mejorar los resultados. Esto ha sido así durante mucho tiempo debido a la *maldición de la dimensionalidad*. Para tareas con vocabularios de varios miles o cientos de miles de palabras el modelo requiere un elevado coste computacional tanto en el aprendizaje como en su uso. Sin embargo, el extenso desarrollo de computadoras con capacidades multiproceso, así como la implementación de instrucciones de tipo vectorial (SSE, SSE2, ...) en los procesadores, ha permitido que el entrenamiento de los modelos pueda llegar a ser muy eficiente. No así la evaluación de los modelos, donde todavía no existen ni herramientas ni sistemas que incorporen estos modelos de forma integrada.

Sin embargo los modelos conexionistas de lenguaje han demostrado ventajas frente al uso de modelos de lenguaje estándar (típicamente N-gramas estadísticos) [Bengio *et al.*, 2001; Schwenk & Gauvain, 2002; Bengio *et al.*, 2003; Castro-Bleda & Prat, 2003; Schwenk *et al.*, 2006; Schwenk, 2007; Bengio, 2008; Schwenk & Koehn, 2008; Schwenk, 2010]. Entre las diversas soluciones presentadas en la literatura a los problemas computacionales de los modelos conexionistas de lenguaje [Morin & Bengio, 2005; Schwenk, 2007; Emami & Mangu, 2007; Park *et al.*, 2010], la más extendida, y exponente común de todos estos trabajos, es relegar el uso del modelo conexionista de lenguaje a una segunda etapa, donde este se aplica sobre un subconjunto de hipótesis elegidas como plausibles por otro sistema. De esta forma la parte más *dura* del proceso de decodificación se lleva a cabo con un modelo de lenguaje más simple, y más eficiente en tiempo, en una primera etapa, mientras que la segunda etapa consiste en procesar una lista de los mejores candidatos (lista de "N-best") o un grafo de palabras que contenga a las N-best, obtenidos por el primer sistema de decodificación, y

se extrae la mejor hipótesis utilizando el modelo conexionista de lenguaje. Este proceso es conocido como "rescoring" o repuntuación de listas de N-best, pero deja algunas cuestiones abiertas que trataremos de responder en este trabajo:

- ¿Sería posible obtener mejores resultados si el sistema fuera completamente integrado? Es decir, ¿mejoraremos los resultados implementando un decodificador que incorpore NN LMs en su proceso de búsqueda?
- Durante la decodificación de una frase aparecen secuencias de palabras que son muy diferentes a las que aparecen en una lista de N-best, ya que estas últimas son *las mejores* entre millones de posibilidades, ¿Será capaz el modelo conexionista de ayudar al sistema a escoger antes las mejores hipótesis permitiendo de alguna manera que el sistema aplique una poda más *dura* en la búsqueda?
- Un objetivo básico de este trabajo es obtener algoritmos basados en redes neuronales que permitan que el entrenamiento y el uso de los NN LMs sean eficientes. ¿Será posible lograr este objetivo manteniendo la calidad del modelo?
- Por último, se muestran diversas aplicaciones que hacen uso de NN LMs para mejorar la calidad de la respuesta del sistema en diferentes tareas relacionadas con la lingüística computacional.

Con esta motivación se desarrollarán dos sistemas, uno de reconocimiento de secuencias, que sirve tanto para escritura manuscrita como para habla, y otro de traducción automática. Investigaremos las ventajas y limitaciones de sistemas totalmente integrados, y mostraremos la mejora obtenida mediante modelos conexionistas de lenguaje cuando se utilizan en estas tareas.

1.2 Sistemas automáticos de reconocimiento/traducción

El objetivo común de un sistema de reconocimiento/traducción es obtener la transcripción de una determinada secuencia de vectores de características en una secuencia de palabras de un determinado vocabulario Ω. Según la tarea que estemos abordando, el sistema puede ser más o menos complejo. Denotaremos con $\bar{x} = x_1 x_2 \dots x_{|\bar{x}|}$ a la secuencia de longitud $|\bar{x}|$ de vectores de características de entrada al sistema, y con $\bar{y} = y_1 y_2 \dots y_{|\bar{y}|}$ a la secuencia de palabras, de longitud $|\bar{y}|$, obtenida como salida por el sistema. Las características específicas de cada sistema son:

- Reconocimiento de secuencias: la entrada es una secuencia de vectores de características extraídos a partir de una secuencia de datos (escritura manuscrita, voz), y la salida es una secuencia de palabras que son la transcripción de la secuencia de entrada. En este caso sabemos que $|\bar{y}| \leq |\bar{x}|$, pero no sabemos a priori cuantas palabras van a ser necesarias para dar una respuesta correcta. El proceso de decodificación es monótono, y podemos diferenciarlo según el problema que resuelve, definiendo el tipo de características de entrada. En este trabajo hemos abordado:
  - Reconocimiento y comprensión automática del habla: la entrada es una secuencia de vectores de características extraídos a partir de una señal de voz.

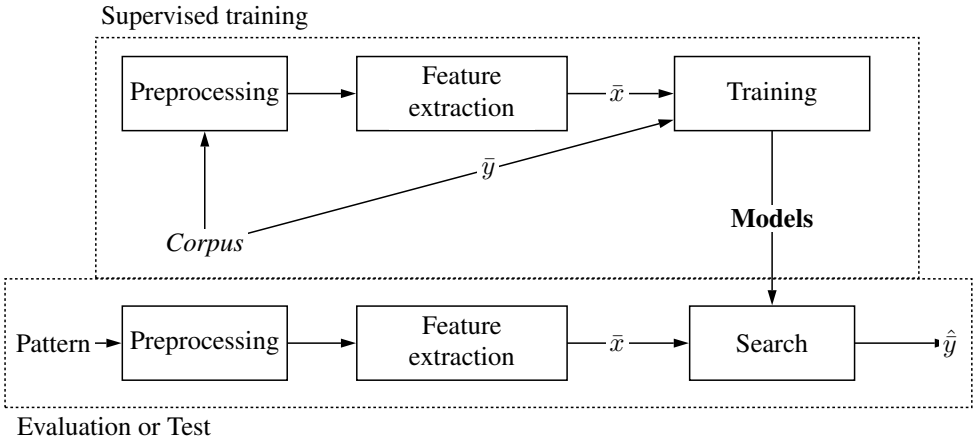


Figura 1.1: *Esquema de las dos fases de un sistema de reconocimiento de formas. //* Pattern recognition system scheme. Referenced at page 289.

- Reconocimiento de escritura manuscrita: los vectores de características de la entrada se extraen a partir de los datos referidos a la escritura manuscrita. Dependiendo de cómo se capturen los datos, tendremos dos modos de funcionamiento:
  - On-line: los vectores de características se extraen a partir de información de posición de un dispositivo o lápiz electrónico que sirve para capturar la escritura.
  - Off-line: los vectores de características se extraen de imágenes de texto tras aplicarles algún tipo de preproceso. Esta es la aproximación seguida en este trabajo.

- Traducción automática: en este caso la entrada al sistema es una secuencia de palabras y la salida es otra secuencia de palabras en una lengua diferente, que es la traducción a dicha lengua de la frase de entrada. Es posible usar voz como entrada y salida del sistema, aunque no será abordado en este trabajo. En este caso $|\bar{x}|$ y $|\bar{y}|$ no tienen ningún tipo de restricción entre ellas, y además el proceso de decodificación no es monótono, esto es, la secuencia de entrada y la de salida se alinean de forma libre, introduciendo un grado de libertad adicional cuya consecuencia es el crecimiento exponencial de la complejidad de la tarea.

Todos estos sistemas están compuestos por dos fases (véase la figura 1.1):

- Entrenamiento supervisado: en esta fase el sistema recibe un corpus etiquetado, de tal forma que para cada entrada se sabe cual debe ser la salida, y se estiman los modelos estadísticos necesarios. Estos servirán durante el proceso de búsqueda para obtener la respuesta ante entradas cuya salida sea desconocida.

- Evaluación: en esta fase el sistema recibe una muestra y produce la salida estadísticamente más probable. Es posible medir el error cometido por el sistema si disponemos de muestras etiquetadas en esta fase.

1.3 Reconocimiento estadístico de formas

La aportación más importante de este trabajo es la utilización de modelos conexionistas en las distintas etapas de los procesos de reconocimiento/traducción, a través de sus ecuaciones básicas como si de un modelo de lenguaje estándar se tratara.

Siguiendo la aproximación estadística, la ecuación fundamental del reconocimiento estadístico de formas [Duda *et al.*, 2001] es:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{y}|\bar{x}) = \tag{1.1}$$

$$= \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y})\,, \tag{1.2}$$

donde tras aplicar la regla de Bayes encontramos los dos modelos básicos que comentamos al principio del capítulo:

- la verosimilitud de la muestra para el modelo utilizado: $p(\bar{x}|\bar{y})$;
- la probabilidad a priori de la hipótesis, dada por el modelo de lenguaje: $p(\bar{y})$.

Esta ecuación se puede generalizar dentro del marco de la Máxima Entropía como una combinación log-lineal de M fuentes de información [Berger *et al.*, 1996a]:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{y}|\bar{x}) = \tag{1.1}$$

$$= \arg\max_{\bar{y}\in\Omega^+} \prod_{m=1}^{M} \mathcal{H}_m(\bar{x},\bar{y})^{\lambda_m}\,, \tag{1.3}$$

donde $\mathcal{H}_m$ es la puntuación (o probabilidad) del modelo m y λ_m es la componente de mezcla de dicho modelo. A partir de esta ecuación se pueden definir todos los sistemas que se presentarán en esta tesis, ajustando de forma adecuada el número de modelos y las componentes λ_m. Bajo esta generalización, en el campo del reconocimiento de secuencias mediante modelos ocultos de Markov, la ecuación está formada por la combinación de tres modelos diferentes: el modelo generativo de Markov, el modelo de lenguaje, y un tercer modelo que penaliza el número de palabras de la hipótesis. Esto da lugar a tres parámetros λ, de los cuales el relativo al modelo generativo se ajusta siempre a $\lambda_0 = 1$, mientras que el modelo de lenguaje y el número de palabras de la hipótesis tienen dos parámetros λ_1 y λ_2, conocidos como "Grammar Scale Factor" (GSF) y "Word Insertion Penalty" (WIP) respectivamente. La ecuación queda de esta manera:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y})^{\lambda_1} exp(|\bar{y}|)^{\lambda_2}\,. \tag{1.4}$$

Desde un punto de vista Bayesiano, es posible generalizar aun más esta ecuación, introduciendo el concepto de función de pérdida, y cambiando la maximización por una minimización [Stolcke *et al.*, 1997]:

$$\hat{\bar{y}} = \arg\min_{\bar{y}\in\Omega^+} \sum_{\bar{y}'\in\Omega^+} p(\bar{y}'|\bar{x}) L(\bar{y},\bar{y}'|\bar{x}) = \quad (1.5)$$

$$= \arg\min_{\bar{y}\in\Omega^+} \sum_{\bar{y}'\in\Omega^+} \prod_{m=1}^{M} \mathcal{H}_m(\bar{x},\bar{y}')^{\lambda_m} L(\bar{y},\bar{y}'|\bar{x})\,, \quad (1.6)$$

donde $L(\bar{y},\bar{y}'|\bar{x})$ es la función de pérdida. Al sumatorio $\sum_{\bar{y}'\in\Omega^+} p(\bar{y}'|\bar{x}) L(\bar{y},\bar{y}'|\bar{x})$ se le denomina riesgo. Si usamos como función de pérdida el complemento de la delta de Kronecker $L(\bar{y},\bar{y}'|\bar{x}) = 1 - \delta(\bar{y},\bar{y}')$, entonces volvemos a tener la ecuación básica del reconocimiento, denominándose criterio de decodificación "Maximum A Posteriori" (MAP). Si definimos la función de pérdida a partir de una medida de disimilitud entre cadenas o secuencias de palabras, entonces se denomina criterio de decodificación "Minimun Bayes Risk" (MBR).

1.4 Arquitectura modular de los sistemas

Los sistemas implementados comparten la filosofía de una arquitectura modular, en lo que denominamos "`dataflow`". Dicho `dataflow` está compuesto por módulos que se comunican entre sí mediante canales de información unidireccionales. En algunos casos la información intercambiada son vectores numéricos (características, probabilidades, ...), o bien información relacionada con la serialización de un grafo en forma de listas de incidencia a través de dichos canales (de izquierda a derecha). Permitiremos que los módulos tengan una conexión que denominaremos de *retroalimentación* (de derecha a izquierda), de tal forma que los módulos de etapas posteriores puedan comunicar la mejor probabilidad que encontró el algoritmo que implementan en el vértice cuya lista de incidencia ha sido totalmente enviada (vértice activo).

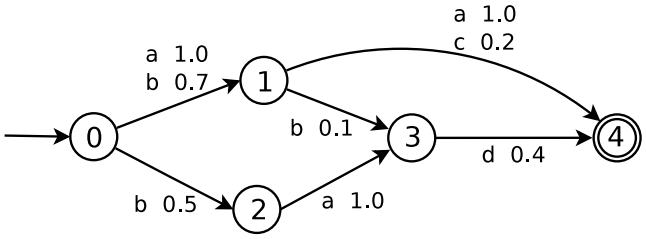
Este protocolo de comunicación en ambas direcciones permite que los algoritmos implementados se sincronicen y puedan utilizar la puntuación del sistema completo para llevar a cabo la poda de la lista de estados activos. Esta poda hará que el algoritmo de búsqueda sea subóptimo, pero la complejidad de los problemas abordados así lo requiere.

En este trabajo se ha tratado de unificar los sistemas presentados, de tal forma que entre ellos se reutilice la mayor cantidad de procesos. Los sistemas son modulares, permitiendo dividir el proceso en las siguientes etapas:

- Generación de hipótesis: esta etapa a su vez puede estar compuesta por diversos procesos consecutivos. El objetivo es generar en orden topológico un grafo de palabras que pueda ser procesado por la etapa siguiente.
- Búsqueda de la hipótesis de mayor probabilidad: esta etapa es común a los sistemas de reconocimiento de voz/escritura, y es ligeramente modificada en traducción. Se permite la incorporación de diferentes modelos que pueden ser combinados linealmente o bien log-linealmente. El proceso de búsqueda se realiza mediante el algoritmo de Viterbi, aplicando tanto poda estática como poda dinámica.

1.4.1. Protocolo para serializar grafos

Los módulos intercambian información de distintos tipos, siendo la más importante la relacionada con grafos. Algunos módulos se encargan de producir dichos grafos, otros de



1: `begin_dag` (`multistage=false`)
2: `vertex` (0)
3: `is_initial` (0)
4: `no_more_in_edges` (0)
5: `vertex` (1)
6: `edge` (0,data=$\{\langle a, 1.0\rangle, \langle b, 0.7\rangle\}$)
7: `no_more_in_edges` (1)
8: `vertex` (2)
9: `edge` (0,data=$\{\langle b, 0.5\rangle\}$)
10: `no_more_in_edges` (2)
11: `no_more_out_edges` (0)
12: `vertex` (3)
13: `edge` (1,data=$\{\langle b, 0.1\rangle\}$)
14: `edge` (2,data=$\{\langle a, 1.0\rangle\}$)
15: `no_more_in_edges` (3)
16: `no_more_out_edges` (2)
17: `vertex` (4)
18: `is_final` (4)
19: `edge` (1,data=$\{\langle a, 1.0\rangle, \langle c, 0.2\rangle\}$)
20: `no_more_out_edges` (1)
21: `edge` (3,data=$\{\langle d, 0.4\rangle\}$)
22: `no_more_out_edges` (3)
23: `no_more_in_edges` (4)
24: `no_more_out_edges` (4)
25: `end_dag` ()

Figura 1.2: *Grafo acíclico dirigido y secuencia de mensajes generada para serializarlo siguiendo el protocolo de incidencia. //* DAG serialization example. Referenced at page 290.

consumirlos/manipularlos, y existe un protocolo que permite serializarlos de manera síncrona. El propósito de este protocolo es doble, por un lado transmitir mensajes que describen dicho grafo, y por otro permitir que los algoritmos se puedan implementar de forma reactiva, es decir, como métodos que dan respuesta a los mensajes del protocolo.

Llamamos a este protocolo *protocolo de incidencia* [España-Boquera *et al.*, 2007] ya que las aristas incidentes sobre un vértice son transmitidas todas juntas. No se trata de un formato de descripción, ni de un formato para representar los grafos en fichero, sino una especificación de un conjunto de mensajes descriptivos y la forma en que deben ser usados. Este protocolo permite encontrar un equilibrio entre la eficiencia de los algoritmos y la simplificación de su implementación debido a la capa de abstracción que ofrece.

El grafo es serializado en orden topológico, lo que permite que cuando llega el mensaje de abrir un vértice nuevo se puedan enviar *todas* las aristas que inciden en él. Por consiguiente, el protocolo sólo permite serializar grafos dirigidos y sin ciclos. Bajo esta restricción encontramos los grafos de palabras utilizados en todos los sistemas de reconocimiento/traducción. El repertorio completo de mensajes se detalla en la tabla 1.1. Para clarificar el protocolo, la figura 1.2 muestra un pequeño grafo y el posible conjunto de mensajes que lo transmiten siguiendo este protocolo.

Tabla 1.1: *Repertorio de mensajes del protocolo de serialización de grafos. //* Graph protocol messages. Referenced at page 290.

begin_dag	It indicates that graph serialization begins.
end_dag	It indicates that graph serialization ends.
vertex	A new vertex is incoming, identified by an integer number. In every `dataflow` algorithm the *current active vertex* refers to the last `vertex` number.
is_initial	The current active vertex is initial with a given probability. This message must be sent *before* of `no_more_in_edges`.
is_final	The current active vertex is final with a given probability.
edge	Every edge is sent using this message. It contains a pairs list of (word, probability) pairs, and an origin vertex. Destination vertex is always the current active vertex (incident edge). Only one message is enough for the serialization of all words between two vertices.
null_edge	It indicates a lambda or null incoming edge. It only contains an origin vertex, and it is useful to simplify some algorithms.
no_more_in_edges	This message is sent after all messages related to the current active vertex. It takes only one argument, the current active vertex identifier. This message means that all possible information about the current active vertex was already received. The module which receives it can start a partial decoding step. At the end of the partial decoding, the receiving module replies the message with a feedback message that contains the best score achieved by any active state.
score_feed_back	This signal is the feedback reply of a module to the `no_more_in_edges` message. It goes in counter-direction of *normal* data flowing. It indicates which is the score of the best active state related to the current vertex.
no_more_out_edges	This message is sent when a vertex will never be an origin vertex of any edge again. It only takes one argument that indicates the vertex identifier number. When a `dataflow` module receives this message, data resources allocated to the corresponding vertex can be freed up.

Tabla 1.2: *Extensión del repertorio de mensajes de la tabla 1.1 para grafos multietapa. //* Multistage (trellis) graph protocol messages. Referenced at page 290.

`begin_dag`	This message is modified, taking one boolean argument that indicates if the graph is multistage (trellis) or not.
`change_stage`	It notifies that all information related to the current stage was sent, and a new stage begins. It is useful because edges with origin and destination in the same stage are forbidden. Therefore, search algorithms can take profit of this constraint.
`edge`	This message is modified at its semantic meaning. Any edges with origin at the current stage are forbidden. Destination vertices are always at the current stage.
`segment_info`	It indicates some segmentation data related to edges. The message takes two arguments, the initial and final segmentation positions. It is useful in machine translation algorithms to send information about how the phrases (or bilingual tuples) are aligned. This information is used to compute the reordering model scores. This message must be sent before its related `edge`.

Extensión a grafos multietapa

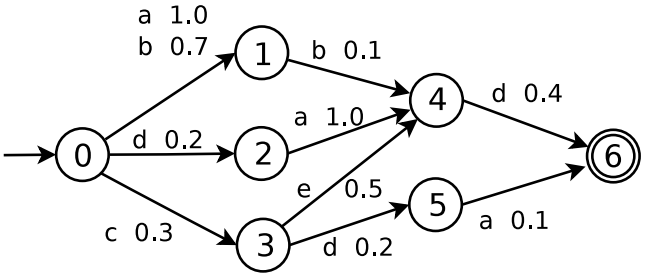
Los grafos multietapa tienen restricciones adicionales que pueden ser tenidas en cuenta a la hora de desarrollar los algoritmos, y por tanto deben ser indicadas en el protocolo de mensajes. Podemos decir que, en un grafo multietapa, cada vértice tiene asociado un número de etapa, y toda arista tiene origen en las etapas anteriores, pero nunca dentro de la misma etapa. El protocolo de incidencia se puede ampliar para incluir esta restricción. La tabla 1.2 muestra dicha extensión y un ejemplo se ilustra en la figura 1.3.

Los grafos multietapa tienen una fuerte relación con el campo de la traducción automática debido a que el reordenamiento de las palabras de la frase de entrada se puede representar mediante un grafo de este tipo.

1.5 Medidas de evaluación

En todas la tareas experimentales presentadas en esta tesis se tienen muestras de referencia que permiten evaluar de forma automática la calidad de los sistemas desarrollados. En función del tipo de tarea las medidas son diferentes. Dado que el tema central del trabajo es el modelado de lenguaje, se puede medir y evaluar su efecto de dos maneras:

- midiendo la perplejidad de los modelos de lenguaje estimados sobre un conjunto de frases de texto, o
- midiendo el efecto del modelo de lenguaje a través del error cometido por el sistema completo de reconocimiento o traducción.



1: `begin_dag` (`multistage=true`)
2: `vertex` (0)
3: `is_initial` (0)
4: `no_more_in_edges` (0)
5: `change_stage`
6: `vertex` (1)
7: `edge` (0,data=$\{\langle a, 1.0\rangle, \langle b, 0.7\rangle\}$)
8: `no_more_in_edges` (1)
9: `vertex` (2)
10: `edge` (0,data=$\{\langle d, 0.2\rangle\}$)
11: `no_more_in_edges` (2)
12: `vertex` (3)
13: `edge` (0,data=$\{\langle c, 0.3\rangle\}$)
14: `no_more_in_edges` (3)
15: `no_more_out_edges` (0)
16: `change_stage`
17: `vertex` (4)
18: `edge` (1,data=$\{\langle b, 0.1\rangle\}$)
19: `no_more_out_edges` (1)
20: `edge` (2,data=$\{\langle a, 1.0\rangle\}$)
21: `no_more_out_edges` (2)
22: `edge` (3,data=$\{\langle e, 0.5\rangle\}$)
23: `no_more_in_edges` (4)
24: `vertex` (5)
25: `edge` (3,data=$\{\langle d, 0.2\rangle\}$)
26: `no_more_out_edges` (3)
27: `edge` (3,data=$\{\langle d, 0.4\rangle\}$)
28: `no_more_out_edges` (3)
29: `no_more_in_edges` (4)
30: `change_stage`
31: `vertex` (6)
32: `is_final` (6)
33: `edge` (4,data=$\{\langle d, 0.4\rangle\}$)
34: `no_more_out_edges` (4)
35: `edge` (5,data=$\{\langle a, 0.1\rangle\}$)
36: `no_more_out_edges` (5)
37: `no_more_out_edges` (4)
38: `end_dag` ()

Figura 1.3: *Grafo acíclico dirigido y multietapa y la secuencia de mensajes generada para serializarlo.* // Multistage DAG serialization example. Referenced at page 290.

Existe una innegable correlación entre ambas medidas, de manera que reducir la perplejidad significa reducir el error cometido por el sistema [Klakow & Peters, 2002]. Sin embargo, no siempre es clara dicha correlación y para poder estimar realmente el impacto del modelo de lenguaje será necesario incorporarlo en la etapa de decodificación. En consecuencia, usaremos la perplejidad sólo como medida orientativa.

En todo caso, para poder calcular estas medidas necesitamos un conjunto de K frases de referencia $\mathcal{R} = \{\bar{r}^{(1)}, \bar{r}^{(2)}, \ldots, \bar{r}^{(K)}\}$, donde cada frase $\bar{r}^{(i)} = r_1^{(i)} r_2^{(i)} \ldots r_{|\bar{r}^{(i)}|}^{(i)}$ es una secuencia de $|\bar{r}^{(i)}|$ palabras de referencia, y el conjunto $\mathcal{Y} = \{\bar{y}^{(1)}, \bar{y}^{(2)}, \ldots, \bar{y}^{(K)}\}$, donde cada frase $\bar{y}^{(i)} = y_1^{(i)} y_2^{(i)} \ldots y_{|\bar{y}^{(i)}|}^{(i)}$ es la secuencia de palabras obtenida como salida del sistema. Podremos calcular el error del sistema midiendo de alguna forma la disimilitud entre la transcripción del sistema $\bar{y}^{(i)}$ y la referencia $\bar{r}^{(i)}$.

1.5.1. Perplejidad

La perplejidad (PPL) [Brown *et al.*, 1992] es una medida muy utilizada en la *teoría de la información*, y se define como la exponencial de la entropía, en este caso normalizada por el número de palabras. Se utiliza para evaluar la bondad de un modelo de lenguaje, calculando la perplejidad de dicho modelo sobre un conjunto de referencia. Dada la referencia $\mathcal{R}$, el modelo de lenguaje permite calcular la probabilidad de la frase $\bar{r}^{(i)}$, $p(\bar{r}^{(i)})$, para todo $i = 1, 2, \ldots, K$. Se puede calcular la entropía del modelo de lenguaje sobre el conjunto de evaluación de la siguiente forma:

$$H(\mathcal{R}) = -\sum_{i=1}^{K} \log p(\bar{r}^{(i)})\,. \tag{1.7}$$

Sea $L = \sum_{i=1}^{K} (|\bar{r}^{(i)}| + 1)$ el número total de palabras en el conjunto (sumando 1 para poder tener en cuenta el fin de frase), la perplejidad se define como:

$$PPL = \exp(\frac{H(\mathcal{R})}{L})\,. \tag{1.8}$$

Esta medida se puede interpretar como el número de confusiones que se producen por cada palabra del conjunto de referencia. De forma equivalente, el valor de perplejidad representa el número de palabras que debería tener un modelo con distribución uniforme para obtener el mismo resultado. Por tanto, se dice que un modelo de lenguaje es mejor que otro si consigue reducir la perplejidad para un mismo conjunto de frases, ya que significa que es capaz de predecir mejor el texto dado.

1.5.2. Word Error Rate

El error a nivel de palabra o "Word Error Rate" (WER) sirve para evaluar la calidad de una transcripción obtenida automáticamente a través de su comparación con una referencia textual. Permite establecer porcentualmente el número de palabras en las que se equivoca el sistema respecto a la referencia. El WER es una medida bien establecida a nivel de frase y de uso extendido en tareas de reconocimiento de voz y escritura.[1] Es una medida derivada de la

[1] En tareas de traducción automática la naturaleza no monótona del problema dificulta la evaluación del sistema mediante esta medida.

distancia de Levenshtein, pero calculándose a nivel de palabra en lugar de a nivel de carácter. Para su cómputo se utiliza un algoritmo de programación dinámica que alinea de forma óptima las palabras de ambas frases, estableciendo el conteo del número mínimo de inserciones, borrados y sustituciones que son necesarios. Dada una transcripción $\bar{y}^{(i)} = y_1^{(i)} y_2^{(i)} \dots y_{|\bar{y}^{(i)}|}^{(i)}$ y su correspondiente referencia $\bar{r}^{(i)} = r_1^{(i)} r_2^{(i)} \dots r_{|\bar{r}^{(i)}|}^{(i)}$, se establece el WER de una única frase con la siguiente ecuación recursiva, resoluble de forma eficiente mediante programación dinámica:

$$WER = \frac{E(|\bar{y}^{(i)}|, |\bar{r}^{(i)}|)}{|\bar{r}^{(i)}|} \cdot 100 = \frac{I+B+S}{|\bar{r}^{(i)}|} \cdot 100\,; \tag{1.9}$$

$$E(j,m) = \begin{cases} 0, & j = 0 \wedge m = 0\,; \\ E(j-1,m) + 1, & j \neq 0 \wedge m = 0\,; \\ \text{mín} \left\{ \begin{array}{l} E(j-1,m) + p_b, \\ E(j,m-1) + p_i, \\ E(j-1,m-1) + p_s \cdot (1 - \delta(y_j^{(i)}, r_m^{(i)})) \end{array} \right\}, & j > 0 \wedge m > 0\,; \end{cases} \tag{1.10}$$

donde $\delta(a,b)$ es la delta de Kronecker, p_b es el coste de un borrado, p_i el coste de una inserción, y p_s el coste de una sustitución. Estableceremos que $p_b = p_i = p_s = 1$. Definida de esta forma, la función $E(|\bar{y}^{(i)}|, |\bar{r}^{(i)}|)$ calcula la suma del número de errores producidos en la transcripción (I inserciones, B borrados y S sustituciones), y el WER es el número medio de errores producidos en la frase. Es posible extender esta medida al conjunto de las K frases y establecer el WER a nivel del conjunto entero. Se interpreta como el porcentaje de palabras, en término medio, mal transcritas por el sistema:

$$WER = \frac{\sum_{i=1}^{K} E(|\bar{y}^{(i)}|, |\bar{r}^{(i)}|)}{\sum_{i=1}^{K} |\bar{r}^{(i)}|} \cdot 100 = \frac{I+B+S}{\sum_{i=1}^{K} |\bar{r}^{(i)}|} \cdot 100\,. \tag{1.11}$$

Esta medida se puede utilizar para evaluar el error a diferentes niveles, dependiendo del tipo de unidad lingüística que tomemos en consideración:

- Si se aplica a nivel de caracteres en lugar de palabras, estamos hablando del "Character Error Rate" (CER), ampliamente utilizado en tareas de reconocimiento escritura manuscrita.
- Si se aplica a nivel de frases, estamos hablando del "Sentence Error Rate" (SER).
- Si se aplica a nivel de conceptos, en una tarea de comprensión del habla, tenemos el "Concept Error Rate" (CER), que no se debe confundir con el "Character Error Rate", a pesar de tener el mismo acrónimo. Ambas se distinguirán según el contexto.

1.5.3. Bilingual Evaluation Understudy

El "Bilingual Evaluation Understudy" (BLEU) es la medida más extendida en traducción automática, introducida por [Papineni *et al.*, 2002]. Consiste en una media geométrica del número de N-gramas (secuencias de N palabras) que comparten la referencia y la transcripción, para valores de $N = 1, 2, 3, 4$. Por su naturaleza, si una de las cuentas es 0, entonces

el resultado de la medida es 0, por ello no se puede aplicar a nivel de frase. Las cuentas se estiman sobre el conjunto entero a evaluar. Sea $\bar{w} = w_1 w_2 \dots w_N \in \Omega^N$ un N-grama cualquiera, y una cadena $\bar{r} = r_1 r_2 \dots r_{|\bar{r}|}$, podemos definir el conteo de apariciones del N-grama $\bar{w}$ en la secuencia $\bar{r}$ como:

$$c_N(\bar{w}, \bar{r}) = \sum_{j=1}^{|\bar{r}|-N+1} \delta(\bar{w}, r_j r_{j+1} \dots r_{j+N-1}) \,. \tag{1.12}$$

Sean los conjuntos $\mathcal{Y}$ de transcripciones dadas por el sistema de traducción, y $\mathcal{R}$ de referencias, con K frases cada uno, descritos igual que antes, y sea

$$S(\{\bar{z}^{(1)}, \dots, \bar{z}^{(K)}\}) \quad = \quad \sum_{i=1}^{K} |\bar{z}^{(i)}| \,,$$

la suma de las tallas de todas las frases de un conjunto dado. El BLEU se define como:

$$BLEU \quad = \quad BP \cdot \sqrt[4]{\prod_{N=1}^{4} \sum_{i=1}^{K} \sum_{\bar{w} \in \Omega^N} \text{mín}(c_N(\bar{w}, \bar{r}^{(i)}), c_N(\bar{w}, \bar{y}^{(i)}))} \,; \tag{1.13}$$

$$BP \quad = \quad \begin{cases} \exp(1 - \frac{S(\mathcal{R})}{S(\mathcal{Y})}), & \text{sii } S(\mathcal{R}) < S(\mathcal{Y}) \,; \\ 1 & \text{sii } S(\mathcal{R}) \geq S(\mathcal{Y}) \,. \end{cases} \tag{1.14}$$

El BLEU es la media geométrica del número de N-gramas que comparte la referencia y la salida del sistema, de manera que permite que se produzcan reordenamientos de secuencias enteras de palabras. Aunque a día de hoy las medidas de evaluación a utilizar en traducción automática son un campo de investigación abierto, el BLEU es la medida escogida como preferida en esta tesis para evaluar los resultados en traducción automática.

1.5.4. Translation Edit Rate

El error de traducción o "Translation Edit Rate" (TER) [Snover *et al.*, 2006] es una modificación del WER que permite introducir reordenamientos entre las palabras, añadiendo una penalización por cada uno de estos reordenamientos. Al igual que el WER, es una extensión de la distancia de Levenshtein, alineando de forma óptima las palabras entre la salida del sistema de traducción y la referencia, calculando inserciones (I), borrados (B), sustituciones simples (S) y distorsiones (D). Se calcula de forma óptima por programación dinámica como sigue:

$$TER \quad = \quad \frac{T(|\bar{y}^{(i)}|, |\bar{r}^{(i)}|, \{1, 2, \dots, |\bar{r}^{(i)}|\}, |\bar{r}^{(i)}|)}{|\bar{r}^{(i)}|} \cdot 100 = \tag{1.15}$$

$$= \quad \frac{I + B + S + D}{|\bar{r}^{(i)}|} \cdot 100 \,; \tag{1.16}$$

$$
\begin{aligned}
T(j,m,\nu,k) &= \begin{cases} 0, & j=0 \wedge m=0\,; \\ T(j-1,m,\nu,k)+p_i, & j\neq 0 \wedge m=0\,; \\ \min\limits_{m'\in\nu} \left\{ \begin{array}{l} T(j-1,m,\nu,k)+p_i, \\ T(j,m-1,\nu-m',m')+p_b+d, \\ T(j-1,m-1,\nu-m',m')+s+d \end{array} \right\}, & j>0 \wedge m>0\,; \end{cases} \\
d &= p_d \cdot (abs(m'-k)-1)\ ; \\
s &= p_s \cdot (1-\delta(y_j^{(i)}, r_{m'}^{(i)}))\,;
\end{aligned}
$$

siendo $\bar{r}^{(i)}$ la referencia i-ésima del conjunto $\mathcal{R}$, $\bar{y}^{(i)}$ la transcripción i-ésima conjunto $\mathcal{Y}$ dada por el sistema, y D el número de distorsiones. De nuevo se utiliza el coste de inserción p_i, borrado p_b, y sustitución p_s, y se añade el coste de la distorsión de una palabra de la referencia p_d. Los valores de dichos costes se toman tal cual funcionan por defecto en la implementación de [Snover *et al.*, 2006]. No obstante, no es factible calcular de forma óptima el TER, ya que el coste del algoritmo es $O(2^{|\bar{r}^{(i)}|}|\bar{y}^{(i)}||\bar{r}^{(i)}|)$. La implementación utilizada en este trabajo introduce heurísticos y búsqueda en haz para reducir el espacio de búsqueda.

1.6 Significancia estadística de los resultados experimentales

Seguiremos la aproximación de [Koehn, 2004] para calcular los intervalos de confianza. De esa manera se establecerá la significancia estadística de los resultados obtenidos.

1.6.1. Intervalo de confianza de las medidas de evaluación

La estimación de intervalos de confianza de las medidas anteriores es una pieza clave a la hora de comparar varios sistemas. En este trabajo calcularemos intervalos con un $95\,\%$ de confianza, lo que permite un ajuste más que suficiente para establecer la significancia estadística de los resultados. Para su cálculo se sigue un proceso estocástico consistente en tomar un conjunto de evaluación con K frases como muestra representativa del universo de posibles muestras, tal y como se explica en [Koehn, 2004]. Cuanto más grande es K, más representativo es el resultado. Se ejecuta un proceso iterativo que se repite $n = 1\,000$ veces, necesarias para poder tener un $95\,\%$ de confianza y que los resultados sean estables. Cada vez se extraen de forma aleatoria y con reemplazamiento K frases del conjunto inicial, lo que quiere decir que pueden aparecer frases repetidas. Se calcula la medida de evaluación sobre dicho conjunto, y se guarda el resultado. Tras repetir este procedimiento n veces, se ordenan los errores cometidos de mayor a menor, se elimina el $2.5\,\%$ inicial y el $2.5\,\%$ final, $5\,\%$ en total, de los extremos de la secuencia ordenada. Para $n = 1\,000$ repeticiones se eliminan 25 por cada extremo. Del resto que queda, se toma el primer y el último resultado, para el ejemplo de $n = 1\,000$ sería el resultado 26 y el 975, y se usan esos valores como puntos extremos del intervalo de confianza. La figura 1.4 ilustra este proceso. Si el intervalo de confianza de dos sistemas así calculado no está solapado, podremos afirmar que la diferencia entre ambos es estadísticamente significativa.

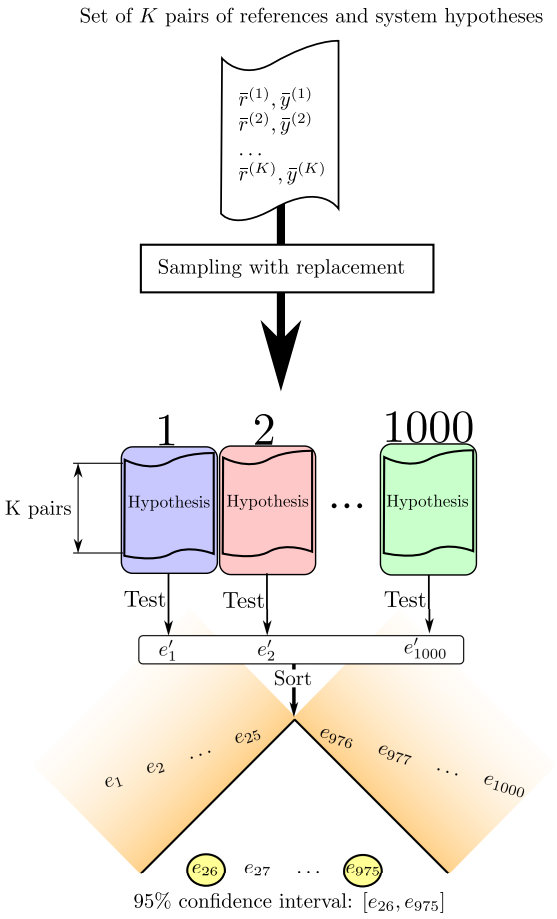


Figura 1.4: *Cálculo de los intervalos de confianza usando* 1 000 *repeticiones.* // Interval confidence computation using 1 000 repetitions. Referenced at page 291.

1.6.2. Comparación entre sistemas: pairwise comparison

En ocasiones sucede que el intervalo de confianza de dos sistemas esté poco solapado. En estos casos se puede ejecutar un nuevo test, conocido como "pairwise comparison", cuyo objetivo es establecer la medida de confianza de la diferencia entre ambos sistemas. Para ello se repite el mismo proceso iterativo que se acaba de explicar, que se repite de nuevo n veces. Si tenemos dos sistemas, A y B, en cada una de las n iteraciones se toman aleatoriamente K frases del conjunto de evaluación. Se calcula la media de la diferencia absoluta entre el error cometido por el sistema A y por el sistema B en cada frase. Tras repetir este proceso n veces, se ordenan las diferencias medidas, y se eliminan los valores de los extremos. Tras esto, se toman el primer y el último elemento del intervalo que son los que definen el intervalo de confianza de la diferencia entre ambos sistemas. Si dicho intervalo es siempre positivo, o negativo, entonces se puede afirmar que un sistema mejora al otro.

1.7 Entrenamiento discriminativo de pesos para combinación de modelos

Como ya se ha introducido, todos los sistemas presentados en este trabajo se pueden describir en términos del marco de Máxima Entropía, de manera que el sistema se entiende como una combinación de diversas fuentes de conocimiento. Bajo esta perspectiva, en traducción automática, es de uso común lo que se conoce como "Minimum Error Rate Training" (MERT) [Och, 2003], procedimiento que permite ajustar los pesos de la combinación log-lineal. Es posible aplicar esta misma idea a sistemas de reconocimiento de voz y escritura, basados en HMMs.

El algoritmo sigue estos pasos:

1. Decodificar el conjunto de desarrollo y generar para cada frase una lista de las N-best, o bien un grafo de palabras que contenga las N-best. En este proceso se utiliza un conjunto de pesos $\bar{\lambda} = \lambda_1 \lambda_2 \dots \lambda_M$.

2. Optimizar la combinación de los pesos de cada uno de los modelos en dichas listas de N-best. La optimización sigue una función objetivo dada, que cambiará según la tarea.[2] Habrá que elegir un algoritmo de optimización de funciones para llevar a cabo este paso:

   - En traducción automática usaremos la implementación estándar del MERT, ofrecida en el "toolkit" Moses [Koehn et al., 2007], y que utiliza el algoritmo de Powell [Powell, 1964].
   - En el caso del reconocimiento de secuencias usaremos una implementación propia del algoritmo MERT usando el algoritmo del Simplex [Nelder & Mead, 1965] para optimizar la combinación de pesos.

   Normalizar el valor de los pesos para la tarea determinada [3]. En este punto el proceso genera un nuevo conjunto de pesos $\bar{\lambda}'$.

[2] En este trabajo sólo usamos WER para reconocimiento de secuencias, y BLEU para traducción automática.

[3] En traducción normalizamos para que la suma de los pesos de la combinación sea exactamente 1, y en reconocimiento de secuencias normalizamos forzando a que el peso del HMM sea 1.

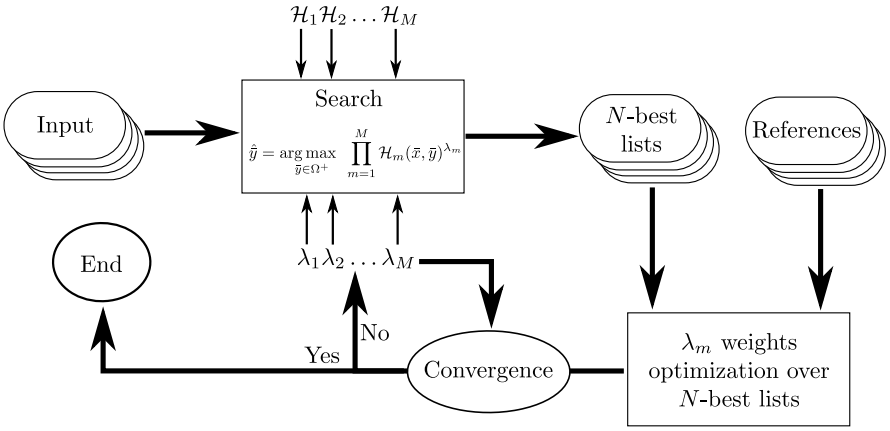


Figura 1.5: *Optimización de los pesos de la combinación log-lineal mediante el algoritmo MERT. //* MERT algorithm. Referenced at page 292.

3. Si la diferencia entre $\bar{\lambda}$ y $\bar{\lambda}'$ es menor que un determinado umbral, parar y devolver $\bar{\lambda}$ como el conjunto de pesos final.

4. Sustituir $\bar{\lambda}$ por $\bar{\lambda}'$.

5. Volver a repetir desde 1).

La figura 1.5 ilustra el funcionamiento del proceso completo. Aprovechando este algoritmo, y que todos los sistemas aquí presentados se pueden entender como una combinación log-lineal de modelos, usaremos esta idea para ajustar los pesos de la combinación de modelos en traducción y también en reconocimiento de voz/escritura.

Esta última idea es una aportación de este trabajo, pues el uso del método presentado de ajuste de parámetros, aunque habitual en la comunidad de traducción automática, no lo es tanto dentro de la comunidad científica que se dedica al reconocimiento de voz o escritura, donde se siguen ajustando los parámetros de los sistemas de reconocimiento (principalmente, los parámetros GSF y WIP en la ecuación (1.4)) utilizando algún método iterativo de prueba y error.

1.8 Objetivos científicos y tecnológicos

Los objetivos que se persiguen en este trabajo se dividen en puramente científicos, aquellos cuyo resultado es un estudio/mejora de determinadas técnicas, y tecnológicos, cuyo resultado es la obtención de herramientas que permitan el desarrollo de las ideas y su investigación futura.

Como objetivos científicos encontramos:

1. Especificación formal de los modelos conexionistas de lenguaje.

2. Especificación de métodos que permitan la incorporación de modelos conexionistas de lenguaje de forma integrada en la etapa de decodificación.

3. Estudio de las ventajas y los inconvenientes de incorporar los modelos conexionistas de lenguaje de forma integrada en la decodificación.

4. Extensión de los NN LMs estándar con un módulo de entrada que lo convierte en un modelo de lenguaje con caché. Dicho módulo constituye un resumen de las interacciones del usuario previas a la actual, de manera que la red neuronal aprende a utilizar esa información para modificar la estimación de las probabilidades del modelo de lenguaje.

En cuanto a los objetivos tecnológicos podemos destacar:

1. Desarrollo e implementación de algoritmos eficientes para el entrenamiento de modelos conexionistas de lenguaje.

2. Desarrollo e implementación de algoritmos eficientes para la evaluación de modelos conexionistas de lenguaje.

3. Desarrollo e implementación de algoritmos eficientes para la utilización de modelos conexionistas de lenguaje en tareas de reconocimiento de secuencias, concretamente en:

   a) habla y

   b) escritura.

4. Desarrollo e implementación de algoritmos eficientes para la utilización de modelos conexionistas de lenguaje en tareas de traducción automática.

5. El uso del algoritmo MERT para establecer el valor de los pesos de la combinación de modelos en reconocimiento del habla/escritura, procedimiento extendido en la comunidad de traducción automática, sin embargo, en la comunidad de reconocimiento de habla o escritura no es tan conocido.

6. Participación en el desarrollo de la herramienta `April`, que incorpora los algoritmos y modelos estudiados en esta y otras tesis del grupo de investigación.

Finalmente, podemos resaltar que, en cuanto a la experimentación en las tareas abordadas, los resultados alcanzados han sido altamente satisfactorios:

- resultados competitivos en una tarea de reconocimiento y compresión automática del habla;
- mejora del estado del arte en reconocimiento de escritura manuscrita;
- resultados al nivel del estado del arte en traducción automática estadística, como se demuestra en el posicionamiento de los sistemas presentados a evaluaciones internacionales (véase el apéndice C).

Parte I

Aportaciones al modelado conexionista de lenguaje

2

MODELOS DE LENGUAJE Y NN LMS

Índice

Un modelo de lenguaje, estimado sobre una lengua determinada, establece la confianza de que un enunciado pertenezca a dicha lengua. Conocer esta confianza permite a las aplicaciones del campo del reconocimiento de formas y del procesamiento del lenguaje natural, entre otras, preferir soluciones lingüísticamente correctas frente aquellas que no lo son. Esto es una información muy valiosa y a día de hoy todos los sistemas la integran.

Hay una gran variedad de técnicas diferentes para estimar esta confianza. Nos interesaremos por aquellas técnicas estadísticas que permitan estimar los modelos de lenguaje de forma automática a partir de ejemplos.

En este capítulo introduciremos las bases de los modelos de lenguaje estadísticos basados en N-gramas, y detallaremos la arquitectura de lo que se conoce como modelo conexionista de lenguaje (NN LM).

Las aportaciones más importantes del capítulo son:

- Especificación de los NN LMs.
- Ampliación y mejora de la arquitectura de los modelos conexionistas de lenguaje.
- Explicación pormenorizada de los problemas computacionales que aparecen al usar o entrenar estos modelos.
- Algunas soluciones a estos problemas planteadas en la literatura, que serán ampliadas en el siguiente capítulo con nuevas propuestas realizadas en este trabajo.

2.1 Modelado de lenguaje

El modelo de lenguaje se introduce en el marco estadístico del reconocimiento de formas a través de su ecuación básica (1.2), que reproducimos aquí:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{y}|\bar{x}) = \tag{1.1}$$

$$= \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y})\,. \tag{1.2}$$

El modelo de lenguaje $p(\bar{y})$, introduce en la búsqueda la probabilidad de que una hipótesis determinada pertenezca al lenguaje de la tarea, de tal forma que permite *castigar* las hipótesis en función de cuan parecidas son a las frases que el modelo aprendió durante su entrenamiento. Según el tipo de técnica utilizada podemos clasificar los modelos de lenguaje en:

- Gramáticas formales: son modelos aceptores que permiten decidir si una hipótesis es o no correcta. Tienen graves problemas cuando las hipótesis tienen errores gramaticales (como sucede en el habla espontánea), debido a que no pueden aceptarlas.
- Modelos estocásticos: son aquellos modelos que en lugar de aceptar o rechazar una construcción, lo que hacen es asignarle una probabilidad. En este grupo vamos a realizar dos clasificaciones:
  - Gramáticas estocásticas: son modelos formales que permiten asignar una determinada masa de probabilidad a una hipótesis concreta. Es necesario aplicar técnicas de suavizado a la masa de probabilidad para que el modelo permita asignar una puntuación a todo el universo de posibles enunciados. Dentro de este conjunto está la gran mayoría de los modelos utilizados actualmente, entre los que destacan los modelos de N-gramas [Jelinek, 1997], formalismo en el que está basado el trabajo que aquí se presenta.
  - Modelos de lenguaje en el espacio continuo: la idea básica de estos modelos es, en lugar de considerar a las palabras símbolos de un alfabeto finito, considerar que son puntos en un espacio métrico continuo, de tal forma que palabras con funciones o significados parecidos dentro de un enunciado estén más cerca en dicho espacio que palabras que no tienen nada que ver una con la otra. El cálculo de la probabilidad que el modelo asocia a una hipótesis se realiza por tanto en dicho espacio continuo. La versión más extendida de este tipo de modelos está basada en los modelos de N-gramas, y se implementa mediante redes neuronales artificiales. Reciben el nombre de "Neural Network Language Models" (NN LMs) [Bengio *et al.*, 2001; Schwenk & Gauvain, 2002; Bengio *et al.*, 2003; Castro-Bleda & Prat, 2003; Schwenk, 2007; Bengio, 2008], y su estudio es el foco principal de este trabajo.

2.1.1. Modelos de lenguaje estadísticos

Los modelos de lenguaje estadísticos estiman la probabilidad $p(\bar{y})$ aplicando la regla de la cadena y descomponiendo su cálculo en el siguiente productorio:

$$p(\bar{y}) = \prod_{j=1}^{|\bar{y}|} p(y_j | y_{j-1} y_{j-2} \dots y_1)\,, \tag{2.1}$$

siendo $\bar{y}$ la secuencia de palabras a estimar, donde cada palabra y_j pertenece al vocabulario Ω. Para llevar a cabo esta descomposición se asume que el lenguaje es un proceso de Markov, y por tanto una palabra únicamente depende de los sucesos anteriores, que definen el estado del sistema. Para poder realizar el cálculo correctamente, se añade una palabra $y_0 = bcc$ (*begin context cue*) y otra $y_{|\bar{y}|+1} = ecc$ (*end context cue*), de manera que bcc identifica el comienzo del enunciado, y ecc indica que ha terminado. Por ejemplo, dado el enunciado $\omega_1\omega_2\omega_3$, su probabilidad se calcularía mediante esta descomposición:

$$p(\omega_1\omega_2\omega_3) = p(\omega_1|bcc) \cdot p(\omega_2|bcc\,\omega_1) \cdot p(\omega_3|bcc\,\omega_1\,\omega_2) \cdot p(ecc|bcc\,\omega_1\,\omega_2\,\omega_3).$$

El número de parámetros necesarios parar poder calcular estas probabilidades condicionales aumenta de forma exponencial con la longitud de las frases. Es por esto que resulta imposible llevar a cabo este cálculo de forma exacta y es necesario simplificar la complejidad del modelo. La simplificación más extendida consiste en limitar el orden del proceso de Markov, de manera que le asignamos un valor N. A esta aproximación del modelo se le denomina modelo de N-gramas. Un modelo de N-gramas calcula la probabilidad condicional de una palabra dadas las $N-1$ anteriores:

$$p(\bar{y}) \approx \prod_{j=1}^{|\bar{y}|} p(y_j | \bar{h}_j)\,, \tag{2.2}$$

donde $\bar{h}_j$ es $y_{j-1}y_{j-2}\dots y_{j-N+1}$ y denota el contexto (o prefijo) del N-grama que termina en la palabra y_j.

Volviendo al ejemplo de $\omega_1\omega_2\omega_3$, si establecemos un orden de $N = 2$ (bigrama), el cómputo de la probabilidad del enunciado se descompondría de esta forma:

$$p(\omega_1\omega_2\omega_3) \approx p(\omega_1|bcc) \cdot p(\omega_2|\omega_1) \cdot p(\omega_3|\omega_2) \cdot p(ecc|\omega_3)\,.$$

La evaluación de la calidad de un modelo de lenguaje se ha definido en la sección 1.5.1 con el concepto de perplejidad. Dado un conjunto de evaluación, es posible medir la perplejidad sobre dicho conjunto para diferentes modelos de lenguaje. Aquel que obtenga el menor valor se considera el mejor modelo para dicho conjunto de evaluación. Si el conjunto de evaluación es suficientemente representativo del dominio en el que estamos trabajando, entonces se puede asumir que dicho modelo es el que mejor se ajusta a nuestra tarea.

2.1.2. Correlación entre PPL y WER

La correlación entre PPL y WER/BLEU es un tema que ha sido investigado por muchos autores [Iyer *et al.*, 1997; Chen *et al.*, 1998; Clarkson & Robinson, 1999; Ito *et al.*, 1999; Printz & Olsen, 2002; Klakow & Peters, 2002]. Concretamente en [Klakow & Peters, 2002] se concluye que existe una fuerte correlación entre PPL y WER, pero no obstante esta relación se oculta por las características del conjunto de test. Por ejemplo, es posible que modelos

de lenguaje con una perplejidad idéntica tengan resultados diferentes al ser usados en el sistema final. No obstante, si la reducción en perplejidad es lo suficientemente significativa, normalmente se observan también mejoras en la calidad del sistema de transcripción.

Para ilustrar esta situación la figura 2.1 muestra la evolución del WER frente a la perplejidad para diferentes modelos de lenguaje sobre el conjunto de validación de la tarea de reconocimiento de escritura manuscrita IAM-DB [Marti & Bunke, 1999] y de la tarea de reconocimiento del habla French Media [Bonneau-Maynard *et al.*, 2005] (véanse los capítulos 8 y 7 respectivamente). Se observa que, a pesar de existir una correlación clara, hay ocasiones en que se rompe dicha tendencia de forma abrupta. En la misma figura se ha representado la reducción en valores porcentuales tomando como referencia la PPL y el WER del sistema con mayor PPL para cada tarea. Se puede observar que existe una tendencia creciente en la mejora, de tal manera que a mejor PPL mejor WER, pero sólo cuando la diferencia en PPL es grande. Además se observa que cada tarea puede tener una tendencia más o menos creciente, probablemente en función de la talla del vocabulario y otros factores que modifican la complejidad de los modelos.

2.2 Modelos de lenguaje de N-gramas

Son el caso más simple y más extendido de modelo de lenguaje estadístico [Jelinek, 1997]. Entre sus deficiencias se encuentra la incapacidad para modelar dependencias a larga distancia. A pesar de su sencillez, conforme crece el valor de N, el número de parámetros necesarios para estimar el modelo crece de manera exponencial. Si tenemos un vocabulario con $|\Omega|$ palabras, el número de posibles N-gramas de talla N es $|\Omega|^N$. Dada la magnitud de este número, es imposible encontrar la cantidad suficiente de datos para estimar el modelo completo. En la literatura se plantean diversas soluciones [Ney & Essen, 1991; Chen & Goodman, 1996], suavizando la estimación de parámetros y permitiendo dar probabilidad a secuencias no vistas durante el entrenamiento. Una alternativa a estas formas de suavizar los modelos es la utilización de NN LMs [Bengio *et al.*, 2003; Bengio, 2008], que permiten interpolar de forma no lineal la probabilidad de secuencias no vistas, como veremos en la sección 2.3.

2.2.1. Estimación de modelos de lenguaje de N-gramas

Los modelos de lenguaje de N-gramas se estiman a partir de un corpus textual a través del conteo de secuencias de palabras, estimando las probabilidades condicionales de la ecuación (2.2). Sea $\bar{k}$ un N-grama de tal forma que $\bar{k} = \bar{q}x$, siendo $\bar{q} \in \Omega^{N-1}$ el prefijo del N-grama $\bar{k}$ y $x \in \Omega$ la última palabra del N-grama. La probabilidad $p(x|\bar{q})$ se estima como:

$$p(x|\bar{q}) = \frac{\mathcal{C}(\bar{q}x)}{\mathcal{C}(\bar{q})}\,, \tag{2.3}$$

donde $\mathcal{C}(\bar{q}x)$ es el número de veces que apareció la secuencia de N palabras $\bar{q}x$ en el corpus de texto, y $\mathcal{C}(\bar{q})$ el número de veces que apareció la secuencia de $N-1$ palabras $\bar{q}$ en el mismo corpus de texto. Nótese que $\mathcal{C}(\bar{q}) = \sum_{\omega\in\Omega} \mathcal{C}(\bar{q}\omega)$.

Estos modelos presentan tres deficiencias básicas:

- Poca capacidad de capturar eventos a gran distancia dado que el número de parámetros crece de forma exponencial con el valor de N.

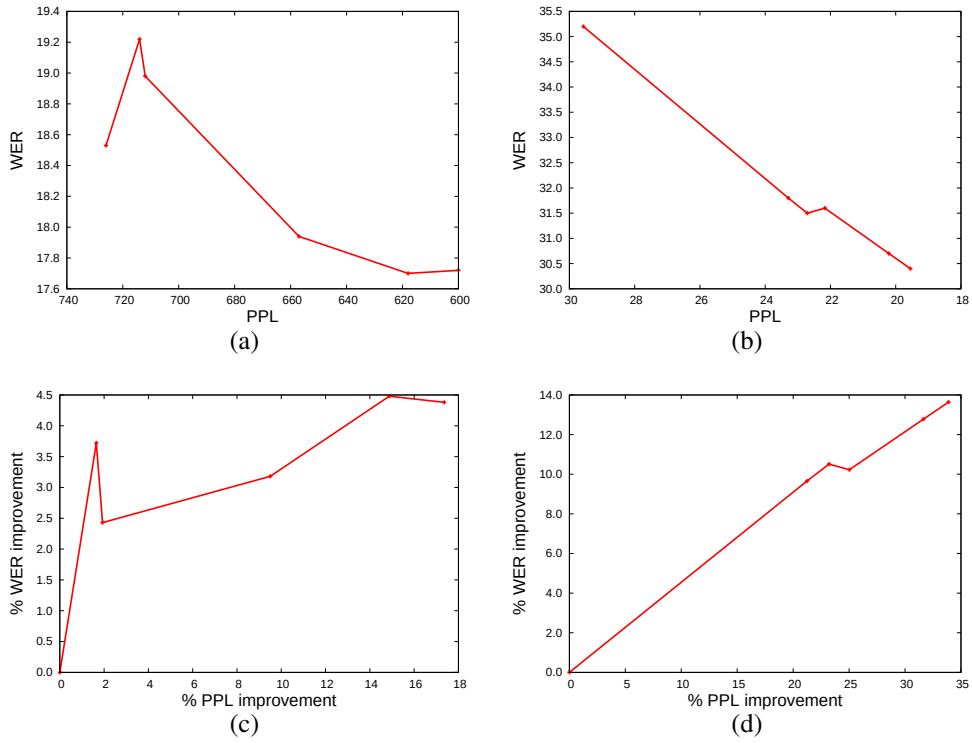


Figura 2.1: *Evolución del WER frente a la PPL para diferentes modelos de lenguaje sobre el conjunto de validación de las tareas de reconocimiento de secuencias a) IAM-DB (escritura) y b) French Media (habla). Debajo, mejora porcentual en WER y en PPL en la tarea c) IAM-DB y d) French Media, tomando como referencia el sistema correspondiente con mayor PPL. Nótese que en a) y b) el eje de abscisas es decreciente (PPL de peor a mejor). //* WER vs PPL evolution for different language models. a) and c) IAM-DB (handwritten recognition); b) and d) French Media (speech recognition). Referenced at page 293.

- Cuando el modelo conoce la secuencia $\bar{q}$, pero se enfrenta a un evento donde $\mathcal{C}(\bar{q}x) = 0$ da como resultado que la probabilidad $p(x|\bar{q})$ es 0, y por tanto, cualquier frase que contuviera dicha secuencia tendría una probabilidad nula. Para resolver este problema veremos que existen diversas formas de restar masa de probabilidad a los eventos vistos y repartirla entre los no vistos.
- La gran mayoría de las secuencias $\bar{q}$ posibles no se encuentran en los datos de entrenamiento, y por tanto no hay información contextual para calcular la probabilidad de N-gramas para esas secuencias si aparecen en el conjunto de test. De nuevo veremos diferentes propuestas de la literatura para solucionar este problema.

2.2.2. Suavizado de las cuentas

Una posible solución al problema de la probabilidad de los eventos no vistos es suavizar la forma de medir las cuentas. Hay numerosas propuestas en la literatura, aquí veremos las más importantes:

Aditivo Es uno de los suavizados más sencillos, y consiste en añadir a todas las cuentas un valor α [Lidstone, 1920; Johnson, 1932; Jeffreys, 1948]. De esa manera la probabilidad se calcula así:

$$p(x|\bar{q}) = \frac{\mathcal{C}(\bar{q}x) + \alpha}{\mathcal{C}(\bar{q}) + \alpha|\Omega|}\,, \tag{2.4}$$

Deleted Trata de estimar las cuentas *reales* usando una partición de control. Separa el corpus en dos conjuntos, el de entrenamiento y el de control. Obtiene las cuentas $\mathcal{C}_t(\bar{k})$ sobre el conjunto de entrenamiento, siendo $\bar{k} = \bar{q}x$ un N-grama de talla N. Calcula el número N_r de N-gramas vistos en entrenamiento con cuenta r, y el número T_r de veces que los N-gramas de cuenta r han sido encontrados en el conjunto de control. A partir de estas cuentas se obtiene la siguiente estimación:

$$p_h(x|\bar{q}) = \frac{T_r}{N_r \mathcal{C}(\bar{q})}\,, \tag{2.5}$$

siendo $\mathcal{C}(\bar{q}x) = r$. Es posible intercambiar ambos conjuntos y combinarlos:

$$p_h(x|\bar{q}) = \frac{T_r^1 + T_r^2}{(N_r^1 + N_r^2)\mathcal{C}(\bar{q})}\,. \tag{2.6}$$

Good-Turing En este caso se trata de establecer el valor de las cuentas r al valor esperado de las cuentas $r^\star$. Dado un N-grama $\bar{q}x$ cuya frecuencia absoluta de aparición es $\mathcal{C}(\bar{q}x) = r$, se utiliza la siguiente fórmula [Good, 1953]:

$$r^\star = (r+1)\frac{N_{r+1}}{N_r}\,, \tag{2.7}$$

$$p(x|\bar{q}) = \frac{r^\star}{\sum_{r=1}^{\infty} r N_r}\,, \tag{2.8}$$

siendo N_r el número de N-gramas que aparecen con cuenta igual a r. Es necesario definir N_0 como el número total de N-gramas diferentes.

Jelinek-Mercer, interpolación de modelos Este tipo de suavizado trata de repartir la probabilidad de los eventos no vistos de manera que aquellos cuyo $N-1$-grama tenga mayor frecuencia, sea más probables. En el caso de bigramas, se interpolan los bigramas con la probabilidad del unigrama, logrando así este comportamiento [Jelinek & Mercer, 1980; Brown *et al.*, 1992]. De forma general se puede definir la interpolación de esta manera:

$$p(x|\bar{q}) = \sum_{i=1}^{N-1} \lambda_i p(x|q_{N-1}q_{N-2}\dots q_i) + \lambda_1 p(x)\,. \tag{2.9}$$

Witten-Bell Tipo de suavizado interpolado recursivo. Su objetivo es tener en cuenta la diversidad de las palabras a la hora de calcular las probabilidades. Veámoslo con un ejemplo. Sean las palabras "spite" y "constant" de forma que:

- Ambas aparecen 993 veces en un determinado corpus.
- Sólo 9 palabras diferentes siguen a "spite", siendo la palabra "of" (979 veces) la más frecuente (debido a la construcción "in spite of").
- 415 palabras diferentes pueden seguir a "constant", siendo la más frecuente (42 veces) "and", pero hay un conjunto de 268 palabras diferentes que sólo le siguen una única vez.

Dadas estas circunstancias, es más probable encontrar nuevos bigramas que empiecen por "constant" que por "spite". Para ello se introduce la siguiente cuenta [Witten & Bell, 1991]:

$$N_{1+}(\bar{q},\cdot) = \sum_{\{x\in\Omega|c(\bar{q},x)>0\}} 1\,, \tag{2.10}$$

que es el número de palabras que siguen a la historia $\bar{q}$ en el conjunto de entrenamiento. Los parámetros de la interpolación se estiman como:

$$\lambda_{\bar{q}} = 1 - \frac{N_{1+}(\bar{q},\cdot)}{N_{1+}(\bar{q},\cdot) + \sum_{x\in\Omega} c(\bar{q},x)}\,, \tag{2.11}$$

y el cálculo del modelo, para una secuencia de palabras tal que $\bar{q} = y_{i-N+1}\dots y_{i-1}$ y $x = y_i$, queda definido de manera recursiva siguiendo esta fórmula:

$$p_{WB}(x|\bar{q}) = \begin{cases} \dfrac{c(x)}{\sum_{x\in\Omega} c(x)} & \text{sii } |\bar{q}| = 0\,; \\ \lambda_{\bar{q}} \dfrac{c(\bar{q},x)}{c(\bar{q})} + (1-\lambda_{\bar{q}})\, p_{WB}(x|q_2\dots q_{N-1}) & \text{sii } |\bar{q}| > 0\,. \end{cases} \tag{2.12}$$

2.2.3. Técnica de back-off

La técnica de "back-off" consiste en dar mayor confianza a las cuentas de N-gramas de mayor orden. Esto resuelve también el problema derivado de los N-gramas $\bar{q}x$ cuyo contexto $\bar{q}$ nunca fue visto en el entrenamiento. Concretamente consiste en aplicar la siguiente formulación:

$$p_{BO}(x|\bar{q}) = \begin{cases} \alpha(\frac{c(\bar{q}x)}{c(\bar{q})}) & \text{sii } c(\bar{q}) > 0\,; \\ d(\bar{q}) \cdot p_{BO}(x|q_2 \dots q_{N-1}) & \text{sii } c(\bar{q}) = 0\,, \end{cases} \tag{2.13}$$

donde $\alpha(\cdot)$ es la función de ajuste de la predicción del modelo y $d(\cdot)$ es la función de descuento. Estas dos funciones nos permiten que la marginalización $\sum_{x\in\Omega} p_{BO}(\bar{q}x)$ sea exactamente 1, y permite que toda posible secuencia de N-gramas tenga una probabilidad asignada. La definición de estas dos funciones será el punto más importante para este tipo de modelos.

Kneser-Ney Así como el descuento Witten-Bell introduce el concepto de la diversidad de las palabras en el modelo, el descuento Kneser-Ney [Kneser & Ney, 1995] tiene en cuenta la diversidad del prefijo de los N-gramas, que explicamos con un ejemplo. Tomamos en consideración la palabra "York":

- Aparece 477 veces en un determinado corpus (es tan frecuente como "foods", "provides", "indicates", entre otras).
- De forma comparativa, el modelo de unigramas le dará una probabilidad alta.
- Sin embargo, casi todas las veces viene sucediendo a la palabra "New" (473 veces).
- La palabra "York" es difícil que aparezca en un bigrama desconocido. El modelo de back-off de unigramas de "York" debe tener poca probabilidad.

Para tener esto en cuenta, el suavizado Kneser-Ney define la cuenta de los prefijos de una palabra:

$$N_{1+}(\cdot x) = \sum_{\{\bar{q}\in\Omega^{N-1}|c(\bar{q})>0\}} 1\,. \tag{2.14}$$

En lugar de estimar los unigramas mediante conteo de palabras, lo hace a través de cuentas de prefijos:

$$p_{KN}(x) = \frac{N_{1+}(\cdot x)}{\sum_{\{x'\in\Omega\}} N_{1+}(x'x)}\,. \tag{2.15}$$

Este descuento se generaliza de manera que es posible aplicarlo de forma recursiva sobre cualquier orden de N.

Kneser-Ney modificado Se generaliza el método de descuento Kneser-Ney entendiendo que el suavizado a aplicar depende de la cuenta del prefijo [Kneser & Ney, 1995]. Se diferencian tres descuentos para el suavizado:

- D_1 si la cuenta es exactamente igual a 1,
- D_2 si la cuenta es exactamente igual a 2,
- D_{3+} si la cuenta es mayor o igual a 3.

2.2.4. Back-off interpolado

El suavizado por back-off tiene problemas con aquellas secuencias de N-gramas que aparecen poco, y el número de estas crece exponencialmente conforme crece el orden N, lo que conlleva a que:

- Si dos N-gramas diferentes con el mismo prefijo ocurren una única vez, ambos tendrán exactamente la misma probabilidad.
- No obstante, uno de ellos puede ser un caso límite, y el otro simplemente estar infravalorado en el conjunto de entrenamiento.

Para remediar este problema se modifica la ecuación (2.13) para combinar siempre las cuentas de distinto orden:

$$p_{IBO}(x|\bar{q}) = \begin{cases} \alpha(\frac{c(\bar{q}x)}{c(\bar{q})}) + d_1(\bar{q}) \cdot p_{IBO}(x|q_2 \dots q_{N-1}) & \text{sii } c(\bar{q}) > 0\,; \\ d_2(\bar{q}) \cdot p_{IBO}(x|q_2 \dots q_{N-1}) & \text{sii } c(\bar{q}) = 0\,, \end{cases} \tag{2.16}$$

de manera que deben ser adaptadas las funciones $\alpha(\cdot)$, $d_1(\cdot)$ y $d_2(\cdot)$ para asegurar que la distribución de probabilidad estimada sea propia. Esta técnica es posible aplicarla al suavizado Kneser-Ney modificado.

2.3 Modelos conexionistas de lenguaje (NN LMs)

Los NN LMs son una aplicación de los modelos de lenguaje basados en la proyección de las palabras en un espacio continuo, asignando a cada palabra una posición en dicho espacio. Matemáticamente un NN LM sigue la misma ecuación que los N-gramas (ecuación (2.2)), pero a diferencia de estos, el modelo es capaz de calcular la probabilidad de todas las palabras del vocabulario dada cualquier posible secuencia de las $N-1$ palabras anteriores. Esto es así gracias a:

- la proyección de las palabras en un espacio continuo, lo que se conoce como codificación distribuida de las palabras;
- y la interpolación realizada por la red neuronal ante secuencias de entrada desconocidas.

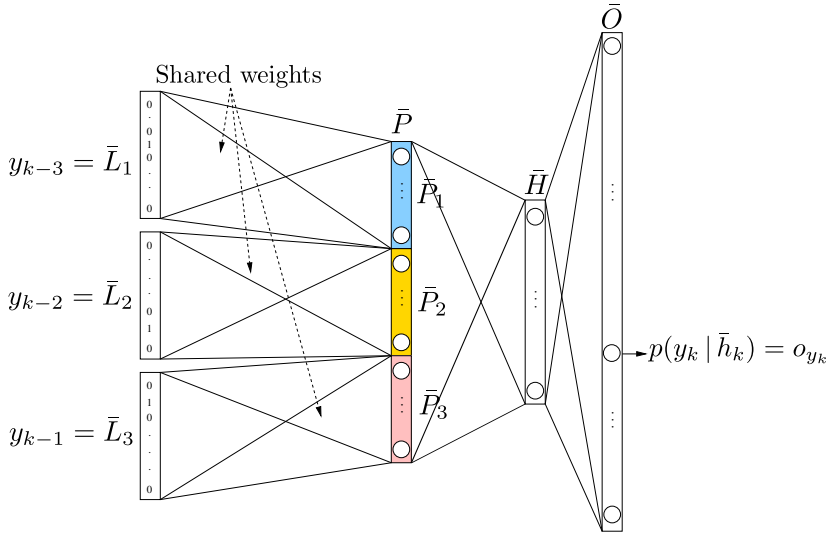


Figura 2.2: *4-grama NN LM durante la fase de entrenamiento, con* $\bar{h}_k = y_{k-3}y_{k-2}y_{k-1}$. // 4-gram NN LM topology during training. Referenced at page 294.

De esta manera el planteamiento teórico de los NN LMs es una posible solución a algunas de las deficiencias de los modelos de N-gramas estándar. No obstante veremos que aparece toda una nueva problemática relacionada con esta forma de modelizar el lenguaje.

La idea del aprendizaje distribuido para representar datos simbólicos, como el caso de las palabras, aparece en el mundo conexionista desde sus primeros días [Hinton, 1986; Elman, 1990; Paccanaro & Hinton, 2000] e igualmente también en el modelado conexionista del lenguaje [Nakamura & Shikano, 1989; Miikkulainen & Dyer, 1991; Castro-Bleda *et al.*, 1999; Xu & Rudnicky, 2000] desde sus inicios. Algunos trabajos presentan la idea de aprender la codificación distribuida de forma desacoplada en un primer paso [Miikkulainen & Dyer, 1991; Tortajada & Castro, 2006]. Fruto de todos estos trabajos, la aproximación a los NN LMs que se ha estandarizado, y que es utilizada en este trabajo, acopla conjuntamente el aprendizaje de la codificación y de las probabilidades [Bengio *et al.*, 2003; Schwenk, 2007].

2.3.1. Arquitectura de los NN LMs

En este trabajo nos hemos basado en la arquitectura de los modelos de [Schwenk, 2007], que detallaremos a continuación, y que se ilustra en la figura 2.2. Dicha arquitectura se compone de cuatro capas de neuronas:

- La entrada $\bar{I}$, que está compuesta por $(N-1)\cdot|\Omega|$ neuronas, donde cada grupo de $|\Omega|$ neuronas representa una palabra, de manera que esas $N-1$ palabras son la parte conocida que condiciona la probabilidad calculada por la red neuronal. A cada conjunto lo denominaremos $\bar{L}_1\bar{L}_2\ldots\bar{L}_{N-1}$, y cada uno de ellos codifica una palabra de forma local, esto es, dada una palabra i en la posición j de la entrada, su codificación $\bar{L}_j$ será un vector con un único *uno* en la neurona de la posición i y *ceros* en las $|\Omega|-1$ neuronas restantes. Esta capa se descarta de la red en la fase de evaluación, ya que es

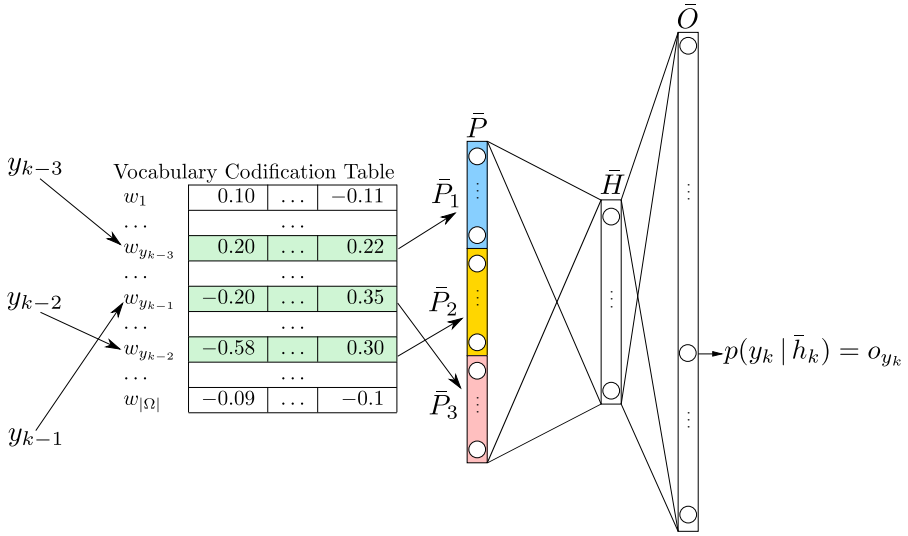


Figura 2.3: 4-*grama NN LM durante la fase de evaluación.* // 4-gram NN LM topology during test. Referenced at page 294.

posible precomputar y almacenar en una tabla la codificación de cada palabra (véase la figura 2.3).

- Cada conjunto $\bar{L}_j$ se conecta con un conjunto $\bar{P}_j$ de neuronas mediante la matriz de pesos $\mathbf{W}_{L,P}$, y el vector de "bias" $\bar{B}_P$. Todos los conjuntos $\bar{P}_j$ tienen idéntica talla. La matriz de pesos y el vector de bias son compartidos por todos los posibles conjuntos de conexiones $\bar{L}_j$ a $\bar{P}_j$. Al conjunto resultante de la concatenación de todos los $\bar{P}_j$ lo denotaremos con $\bar{P}$, y será la capa de proyección. Su función es codificar/proyectar cada una de las $N-1$ palabras en un espacio continuo, de manera que permita a palabras parecidas tener una codificación parecida. La probabilidad condicional del modelo de lenguaje se computa sobre este espacio, permitiendo que el suavizado de los eventos no vistos en el entrenamiento esté basado en la semejanza entre las palabras. La función de activación de la capa $\bar{P}$ es la lineal.

- La capa de proyección se conecta con la capa oculta, formada por el conjunto de neuronas $\bar{H}$, mediante la matriz de pesos $\mathbf{W}_{P,H}$ y el vector de bias $\bar{B}_H$, y su función de activación será la tangente hiperbólica.

- Finalmente la capa oculta se conecta con la capa de salida, formada por $|\Omega|$ neuronas, y que denominaremos $\bar{O}$. La conexión se realiza a través de la matriz $\mathbf{W}_{H,O}$ y el vector bias $\bar{B}_O$. La función de activación de la capa de salida es la "softmax", que permite asegurar que la salida de la red son probabilidades [Bishop, 1995].

Usando toda esta notación podemos escribir el cómputo realizado por la red neuronal de esta manera:

$$\bar{P} = \bigcup_{i=1}^{N-1} \left(\bar{L}_i \cdot \mathbf{W}_{L,P}^T + \bar{B}_P\right) , \quad (2.17)$$

$$\bar{H} = tanh(\bar{P} \cdot \mathbf{W}_{P,H}^T + \bar{B}_H) , \quad (2.18)$$

$$\bar{A} = \bar{H} \cdot \mathbf{W}_{H,O}^T + \bar{B}_O , \quad (2.19)$$

$$\bar{O} = \frac{exp(\bar{A})}{\sum_{i=1}^{|A|} exp(a_i)} , \quad (2.20)$$

donde a_i se refiere a la componente i del vector $\bar{A}$. Vamos a destacar que el coste de la ecuación (2.17) se puede reducir drásticamente teniendo en cuenta que cada entrada $\bar{L}_i$ está codificada localmente. Dada esta característica el resultado de la multiplicación de $\bar{L}_i \cdot \mathbf{W}_{L,P}^T$ es escoger la columna de $\mathbf{W}_{L,P}$ correspondiente a la única neurona activada en $\bar{L}_i$. Por tanto, si el coste original es $O(|\bar{L}_i| \cdot |\bar{P}_i|)$, este se reduce a $O(|\bar{P}_i|)$. Hay que tener en cuenta que normalmente $|\bar{L}_i| \approx |\Omega|$, y que $|\bar{P}_i| << |\bar{L}_i|$.

En la literatura, el vector de bias $\bar{B}_P$ suele ignorarse, no entrenándolo y no teniéndolo en cuenta en los cálculos. No obstante en este trabajo sí lo tendremos en cuenta.

La red neuronal se entrena mediante el algoritmo de retropropagación del error o "Error Back-Propagation" (BP) [Rumelhart *et al.*, 1986] en su modo estocástico y "on-line", utilizando como función de error la entropía cruzada, que se define como sigue:

$$e_j = -\sum_{i=1}^{|\bar{O}|} t_i \log(o_i) , \quad (2.21)$$

siendo t_i la salida deseada para la neurona i de la capa de salida, o_i la salida calculada por la red para la neurona i, y e_j el error calculado para la muestra de entrenamiento j. El error cometido por la red en un conjunto de R muestras se mide sumando todos los errores y después dividiendo por el número de muestras:

$$E = -\frac{1}{R} \sum_{j=1}^{R} \sum_{i=1}^{|\bar{O}|} t_i^{(j)} \log(o_i^{(j)}) , \quad (2.22)$$

donde $t_i^{(j)}$ es la salida deseada para la neurona i en la muestra j, y $o_i^{(j)}$ el valor de la salida i de la red neuronal para la muestra j. El entrenamiento de la red es on-line, pero el error se estima sobre un subconjunto de muestras de entrenamiento/validación [Schwenk, 2007].

Un gran problema de las redes neuronales es el sobre-entrenamiento, y por tanto, la falta de generalización cuando ven datos diferentes de los aprendidos. Para mitigar este problema incorporamos al algoritmo el término de regularización L_2 denominado "weight decay", que extiende la función de error añadiéndole un ϵ de la suma del cuadrado de los pesos de la red neuronal:

$$e_j = -\sum_{i=1}^{|\bar{O}|} t_i \log(o_i) + \epsilon \sum_{w \in \bar{W}^\circ} \frac{w^2}{2} , \quad (2.23)$$

donde $\bar{W}^\circ = \mathbf{W}_{L,P} \cup \mathbf{W}_{P,H} \cup \mathbf{W}_{H,O}$. Este término tratará de hacer que los pesos tengan valores pequeños, permitiendo suavizar y por tanto generalizar más. Habitualmente se evita

aplicar el término de regularización a los pesos bias, permitiendo que estos se ajusten en función del valor del resto de pesos.

Decimos que el entrenamiento es *estocástico* porque en cada época, en lugar de mostrar a la red todo el conjunto de entrenamiento, se entrena con un subconjunto escogido de forma aleatoria y con reemplazamiento. Esto permite entrenar la red sólo con un subconjunto de los datos disponibles, más todavía cuando el tamaño del corpus es de varios cientos de millones de palabras. Además, en muchos casos basta con una pequeña cantidad de muestras (comparada con el conjunto completo) para que la red llegue a un punto de convergencia. El algoritmo 2.1 describe el método de entrenamiento seguido en este trabajo.

Algoritmo 2.1 Entrena una red neuronal con reemplazamiento y tomando datos de diferentes corpus con diferente probabilidad. *// Neural network training with replacement and different a priori distribution corpora. Referenced at page 295.*

Require: A set $c_1, c_2, \ldots, c_R$ of dataset patterns from R different corpora, a priori probability for each dataset $p_1, p_2, \ldots, p_R$, and a replacement size

Ensure: Train a NN LM using replacement and following the probability distribution of each dataset, and returns the cross-entropy error

1: **function** TrainWithReplacementAndDistribution($Repl, \langle c_1, p_1\rangle, \ldots, \langle c_R, p_R\rangle$)
2: **begin**
3: Let $e = 0$ the error produced by the NN LM
4: **for** $i = 1, \ldots, Repl$ **do**
5: Let $j =$ corpora index from 1 to R sampled using distribution probability $p_1, \ldots, p_R$
6: Let $y_{k-N+1} \ldots y_k =$ random N-gram sampled from corpora dataset c_j
7: Let $\bar{I} = \bigcup_{l=k-N+1}^{k-1}$ local encoding of word y_l
8: Let $\bar{T} =$ local encoding of word y_k
9: Take $\bar{I}$ as input and $\bar{T}$ as desired or target output to train the NN LM
10: Compute NN LM forward step
11: Let $e' = \sum_{i=1}^{|\bar{O}|} t_i \log(o_i)$, being t_i the i-th element of $\bar{T}$, and o_i the i-th element of $\bar{O}$ // Cross-entropy computation
12: $e = e - e'$
13: Update NN LM weights
14: **end for**
15: **return** $e/Repl$
16: **end function**

El entrenamiento de la red neuronal se puede resumir con estas operaciones:

$$\frac{\partial E}{\partial O} = \left\langle o_1 - t_1, o_2 - t_2, \ldots, o_{|\bar{O}|} - t_{|\bar{O}|} \right\rangle , \tag{2.24}$$

$$\bar{B}_O = \bar{B}_O + \eta \frac{\partial E}{\partial O} , \tag{2.25}$$

$$\mathbf{W}_{H,O} = \epsilon \mathbf{W}_{H,O} + \eta \bar{H}^T \frac{\partial E}{\partial O} , \tag{2.26}$$

$$\delta \bar{H} = \mathbf{W}_{H,O} \frac{\partial E}{\partial O} , \tag{2.27}$$

$$\frac{\partial E}{\partial H} = \left\langle \delta\bar{h}_1 \cdot 0.5(1-\bar{h}_1^2), \ldots, \delta\bar{h}_i \cdot 0.5(1-\bar{h}_i^2), \ldots \right\rangle , \tag{2.28}$$

$$\bar{B}_H = \bar{B}_H + \eta \cdot \frac{\partial E}{\partial H} , \tag{2.29}$$

$$\mathbf{W}_{P,H} = \epsilon \mathbf{W}_{P,H} + \eta \cdot \bar{P}^T \cdot \frac{\partial E}{\partial H} , \tag{2.30}$$

$$\frac{\partial E}{\partial P} = \mathbf{W}_{P,H} \cdot \frac{\partial E}{\partial H} , \tag{2.31}$$

$$\bar{B}_P = \bar{B}_P + \frac{1}{N-1} \sum_{i=1}^{N-1} \eta \frac{\partial E}{\partial P_i} , \tag{2.32}$$

$$\mathbf{W}_{L,P} = \epsilon \mathbf{W}_{L,P} + \frac{1}{N-1} \sum_{i=1}^{N-1} \eta \bar{L_i}^T \frac{\partial E}{\partial P_i} . \tag{2.33}$$

En el apéndice B se explica el procedimiento de entrenamiento para Redes Neuronales Artificiales ("Artificial Neural Networks" o ANNs) generales.

2.3.2. NNLMs en la literatura

A pesar de la expansión que está teniendo el uso de NN LMs en los campos de reconocimiento de voz/escritura y traducción automática, muy pocas implementaciones se encuentran disponibles de forma libre. La mayoría de las publicaciones existentes siguen el trabajo de [Schwenk, 2007].[1] Las diferencias entre el NN LM presentado en dicho trabajo y el aquí propuesto son básicamente las siguientes:

- El uso del vector de pesos bias $\bar{B}_P$ en la capa de proyección.
- La inclusión del parámetro de regularización en la capa de proyección.

La inclusión o no del bias en la capa de proyección se traduce en una forma ligeramente distinta de obtener la codificación distribuida de una palabra. Dada la palabra cuya codificación local activa la posición m, su codificación distribuida sería en cada caso:

a) Sin usar bias en la capa de proyección:

$$\bar{P}_j = \begin{pmatrix} w_{1,m} \\ w_{2,m} \\ \ldots \\ w_{|P_j|,m} \end{pmatrix}^T \tag{2.34}$$

b) Usando bias en la capa de proyección:

$$\bar{P}_j = \begin{pmatrix} w_{1,m} + b_1 \\ w_{2,m} + b_2 \\ \ldots \\ w_{|P_j|,m} + b_{|P_j|} \end{pmatrix}^T \tag{2.35}$$

[1] El CSLM toolkit de H. Schwenk permite entrenar NN LMs y está disponible en la dirección web `http://www-lium.univ-lemans.fr/cslm/`.

siendo $w_{i,m}$ la posición fila i, columna m, de la matriz de pesos $\mathbf{W}_{L,P}$, y b_i la posición i del vector de bias $\bar{B}_P$.

Relación entre el weight decay y el bias

En el campo del aprendizaje automático, los términos de regularización como el weight decay no se aplican nunca a los pesos bias de las neuronas [Duda *et al.*, 2001]. De esta manera, se fuerza a que el resto de pesos tenga valores cercanos a cero, permitiendo que el bias se ajuste en función de las muestras.

Dado este escenario, si usamos el weight decay en la capa de proyección, los pesos no bias correspondientes a las palabras con frecuencia de aparición baja podrían llegar a tener valor cero. Esto implica que es necesario utilizar el peso bias para evitar que la codificación de estas palabras sea cero. Por lo tanto, existe una íntima relación entre el uso del weight decay en la capa de proyección y el uso del bias para evitar palabras con codificación cero.

Esta exposición abre una discusión muy interesante. De manera sistemática, como ya se ha comentado, el weight decay se suele eliminar de la capa de proyección, del mismo modo que también se elimina el bias en dicha capa. De esta forma, cuando termina el entrenamiento, las palabras poco frecuentes tienen una codificación que ha cambiado muy poco (o nada) respecto a la codificación que se fijó al inicio. Dado que habitualmente los pesos se inicializan de forma aleatoria, las palabras poco frecuentes tendrán codificaciones cuasi-aleatorias y diferentes entre sí (véase la ecuación (2.34)).

Por otro lado, si utilizamos weight decay y añadimos el bias a la capa de proyección, los pesos no bias tenderán a cero, y la codificación de esas palabras poco frecuentes coincidirá con el valor del vector bias (véase la ecuación (2.35)). En consecuencia, la codificación de todas ellas será muy parecida y se lograrán mejores resultados.

Dada esta discusión, la única razón a priori para evitar el uso del weight decay y del bias en la capa de proyección es acelerar el entrenamiento. Evitar el weight decay puede reducir el coste en un factor lineal con $|\Omega|$, la talla del vocabulario, coste despreciable comparado con el coste del cómputo de la capa de salida. Por tanto, asumiendo que a nivel computacional ambas aproximaciones son parecidas, no se conoce ninguna razón matemática por la que no utilizar este parámetro. Es más, este factor puede ser útil para que las palabras poco frecuentes tengan pesos pequeños, y como acabamos de exponer, una codificación más parecida entre sí. Retomaremos este problema en la sección 3.1.1.

Para finalizar este apartado, comentaremos que, tanto la formalización como la implementación de las redes utilizadas en este trabajo siguen la formulación original de redes neuronales, con lo que se abre la posibilidad de en un futuro poder extender la capa de proyección permitiendo utilizar más de una capa de neuronas, o utilizar otro tipo de funciones de activación, no sólo la lineal.

2.4 Combinación de NN LMs y N-gramas estándar

Trabajaremos normalmente con NN LMs que son el resultado de combinar linealmente la salida de varias redes neuronales, con diferentes tamaños en la capa de proyección, lo que nos asegurará obtener una mayor generalización [Schwenk, 2007].

Al mismo tiempo combinaremos linealmente o log-linealmente la probabilidad computada por las redes neuronales con la de un modelo estadístico estándar de N-gramas. Inde-

pendientemente de la estrategia escogida, la combinación de ambos modelos es un requisito clave para mejorar los resultados [Schwenk & Koehn, 2008].

Según la tarea del experimento se escogerá la forma en que serán combinados los modelos. Si combinamos K redes neuronales, podemos denotar con $p_k(y_j|\bar{h}_j)$ para $k = 1, 2, \ldots, K$ la probabilidad calculada con el modelo conexionista k, y con $p_0(y_j|\bar{h}_j)$ la probabilidad del modelo de N-gramas estándar. Este último es estimado sobre el vocabulario completo de la tarea, y será un modelo interpolado mediante el suavizado de Kneser-Ney modificado. La combinación de los modelos conexionistas de lenguaje con el estadístico de N-gramas estándar puede ser lineal o log-lineal:

- Combinación lineal: en este caso se consideran todas las redes neuronales más el modelo de N-gramas estándar para calcular los pesos α_k, minimizando la PPL sobre un conjunto de validación:

$$p(y_j|\bar{h}_j) = \sum_{k=0}^{K} \alpha_k p_k(y_j|\bar{h}_j)\,, \tag{2.36}$$

  con $\sum_k \alpha_k = 1$ y $0 \leq \alpha_k \leq 1$, siendo α_0 el factor de combinación del modelo de N-gramas estadístico.

- Combinación log-lineal: en este caso combinaremos primero linealmente todas las redes neuronales, minimizando la PPL sobre un conjunto de validación, y más tarde combinaremos log-linealmente el NN LM, resultante de la combinación lineal, con el N-grama estándar, maximizando el BLEU. Este caso será aplicado sólo a las tareas de traducción, si bien sería posible aplicarlo a tareas de reconocimiento minimizando el WER. Denotaremos con λ_0 y λ_1 los factores de combinación log-lineal del modelo de N-gramas estándar y del modelo conexionista respectivamente. Esta combinación se integra dentro del marco de máxima entropía, siguiendo la ecuación (1.3):

$$p(y_j|\bar{h}_j) = p_0(y_j|\bar{h}_j)^{\lambda_0} \left(\overbrace{\sum_{k=1}^{K} \alpha_k p_k(y_j|\bar{h}_j)}^{\text{combinación lineal}} \right)^{\lambda_1} . \tag{2.37}$$

En [Schwenk & Koehn, 2008] se comparan ambas formas de combinar los modelos, concluyendo que no hay diferencias significativas entre ellas. En nuestro trabajo, para reconocimiento de habla y de escritura usaremos la combinación lineal, por ser más consistente con la ecuación básica del reconocimiento (1.2), si bien en traducción automática se utilizará la combinación log-lineal debido a que la ecuación (1.3) está mucho más extendida en ese campo.

2.5 Deficiencias del modelo NN LM

Una de las grandes aportaciones de los NN LMs al modelado del lenguaje es la codificación distribuida de las palabras de entrada gracias a la capa de proyección. Esta codificación, además, es independiente de la posición en que se encuentre la palabra en la entrada de la

red. Sin embargo, dicha proyección no está libre de problemas, conocidos y constatados por otros autores [Hai-Son *et al.*, 2010]. El mayor problema es la codificación de aquellas palabras que aparecen rara vez en el texto, y se vuelve un problema grave con aquellas palabras que aparecen una única vez. Algunas de estas palabras, conforme crece el tamaño del corpus, ni tan siquiera llegan a ser vistas durante el entrenamiento, y otras muchas de ellas se presentan muy pocas veces[2]. Esto desenlaza en que su codificación es pobre, muy parecida al punto aleatorio inicial dado por la inicialización de dichas conexiones. Este problema tiene más importancia todavía cuando se utilizan modelos que evitan la aplicación del parámetro de regularización a la capa de proyección (véase sección 2.3.2). Aumentando la calidad de dicha proyección se puede a su vez mejorar la distribución de probabilidad, y por tanto el funcionamiento del modelo. En la sección 3.1.1 se plantean las soluciones encontradas en la literatura, y se aportan nuevas ideas como trabajo futuro a desarrollar.

El otro de los grandes problemas es el coste computacional que requiere utilizar NN LMs. Este coste está gobernado por el tamaño de la salida de la red neuronal, debido a que es necesario calcular el factor de normalización de la función de activación softmax, denominador de la ecuación (2.20). En la fase de entrenamiento, este denominador siempre se debe calcular, pero en la fase de evaluación, para una misma entrada de la red neuronal, tan sólo es necesario calcular la probabilidad de unas pocas palabras. Sin embargo, este denominador obliga a calcularlas todas. Resumiremos las propuestas más interesantes de la literatura al respecto de este tema en la sección 3.1.2. También se encontrará en dicha sección la formalización de la solución seguida en este trabajo. El capítulo 4 aporta una idea original para reducir el coste computacional en la fase de evaluación.

2.6 Últimas tendencias en modelado conexionista de lenguaje

En los últimos años el uso de modelos conexionistas de lenguaje ha experimentado un importante incremento. Destacaremos que en todos los casos la aproximación ampliamente aceptada es la basada en dos fases:

1. primero reconocer con un N-grama estándar y
2. después realizar un rescoring de listas de N-best mediante NN LMs.

En [Mnih & Hinton, 2007] se presentan tres modelos de lenguaje basados en los últimos de trabajos de [Bengio *et al.*, 2006] con "Restricted Boltzmann Machines" (RBMs). Presentan estas tres aproximaciones:

- Factored RBM Language Model: un modelo probabilístico de secuencias de palabras que utiliza codificación distribuida para las palabras y captura dependencias entre ellas a través de variables ocultas estocásticas. Gracias al uso de RBMs consiguen aumentar el número de variables ocultas a utilizar. El entrenamiento se realiza de forma eficiente a través del "constrastive divergence learning" [Hinton, 2000].
- Temporal Factored RBM: amplía el modelo anterior añadiendo conexiones recursivas entre las capas ocultas de la RBM. De esa manera funcionan a modo de red recurrente, permitiendo que las dependencias entre palabras puedan ser de longitud variable.

[2]Debido al uso de algoritmos que seleccionan las muestras con reemplazamiento, no podemos asegurar que todas las palabras del corpus se utilicen en el entrenamiento.

- Log-Bilinear Language Model (LBLM): este modelo parametriza de forma directa la distribución de probabilidad de la siguiente palabra dadas las anteriores, evitando introducir variables estocásticas ocultas. Esta parametrización sustituye a las variables binarias estocásticas utilizadas por los otros dos modelos. Obtienen sus mejores resultados en PPL usando este el modelo, aunque no dan resultados de reconocimiento.

El citado trabajo [Mnih & Hinton, 2007] presenta experimentos con el corpus AP News [Bengio *et al.*, 2003], obteniendo con el mejor de los modelos presentados una perplejidad en el conjunto de test de 92.1. En trabajos previos de [Bengio *et al.*, 2003] el mejor resultado presentado fue de 109 puntos de perplejidad.[3]

Los mismos autores en [Mnih & Hinton, 2008] mejoraron los modelos LBLM introduciendo una estructura jerárquica en la salida de los mismos. De nuevo tenemos resultados de perplejidad en el test del corpus AP News, pero en esta ocasión no se proporcionan resultados de combinar los modelos presentados con un modelo estándar de N-gramas. Presentan una perplejidad de 112 puntos para el mejor modelo, frente a los 117 que tenían antes aplicar la salida jerárquica. Sin embargo, en cuestión de coste computacional, los modelos jerárquicos (HLBLM) necesitan 200 veces menos tiempo de entrenamiento, necesitando tan sólo 32 minutos por época de entrenamiento.

En [Hai-Son *et al.*, 2010] presentan resultados comparando LBLMs con NN LMs, y al mismo tiempo plantean tres maneras alternativas de entrenar la capa de proyección para mejorar los resultados de los NN LMs. Muestran resultados en la tarea NIST Árabe-Inglés del 2009 de traducción automática. Los modelos gráficos, tipo LBLM, no logran mejorar al NN LM ni en perplejidad ni en BLEU. Utilizando un vocabulario de las 10K palabras más frecuentes, los modelos gráficos obtienen 38.2 puntos de BLEU frente a 38.3 del NN LM, que llega hasta 38.4 mejorando la capa de proyección. Usando el vocabulario completo de la tarea, presentan un resultado de 38.6 puntos de BLEU mediante NN LMs, que mejoran hasta 38.7 modificando el entrenamiento de la capa de proyección. El sistema de traducción sin usar NN LMs obtiene 37.8 puntos de BLEU. Los resultados presentan una mejora de 0.9 puntos de BLEU sobre este baseline, dejando abiertas ciertas dudas sobre las mejoras obtenidas mediante los modelos gráficos mencionados. Presentan una mejora de la capa de proyección que consiste en inicializar la red de manera que la proyección de todas las palabras del vocabulario sea exactamente la misma (escogida aleatoriamente).

En [Hai-Son *et al.*, 2011] se presenta una modificación del NN LM estándar que permite estructurar la salida de la red neuronal en varios grupos de neuronas, inspirada en los HLBLMs de [Mnih & Hinton, 2008]. Cada uno de estos grupos representa diferentes distribuciones de las palabras en clases, además de un grupo que es equivalente a la salida de un NN LM estándar. Con estas modificaciones consiguen obtener una mejora del 23 % de perplejidad y del 9 % de CER (Character Error Rate). La tarea es del proyecto *Gale*, reconocimiento automático del habla, escogiendo el Chino Mandarín como lengua.

En otra línea podemos encontrar los trabajos de [Mikolov *et al.*, 2010, 2011] con modelos de lenguaje basados en redes neuronales recurrentes. Presentan resultados con el corpus de habla "Wall Street Journal", logrando bajar desde 221 puntos de PPL del baseline con N-gramas estándar, hasta 121 usando redes recurrentes combinadas con el N-grama estándar, y la misma mejora se observa en WER, desde 17.2 % de WER para el baseline usando N-gramas, hasta 15.5 % de WER para el modelo recurrente. En esta línea prometedora sería

[3] A modo comparativo hemos entrenado NN LMs estándar usando el mismo corpus alcanzando una perplejidad 95.6.

posible alimentar el rescoring de listas de N-best que hacen las redes recurrentes mediante una lista generada con un sistema integrado de NN LMs, y combinar los NN LMs con la red recurrente, ya que no es posible obtener el sistema integrado con la red recurrente por motivos de eficiencia.

También con redes recurrentes, en este caso del tipo "Long-Short Term Memory" (LSTM), en [Frinken *et al.*, 2012] hemos publicado en nuestro grupo de investigación, en colaboración con el grupo del "Institue of Applied Mathematics" (IAM) de la Universidad de Bern, resultados de modelado de lenguaje para la tarea IAM-DB de reconocimiento de escritura manuscrita "off-line". Los resultados son preeliminares y la tarea no es la más adecuada para hacer uso de modelos recurrentes, pero no obstante este trabajo ha servido para validar el uso de las LSTMs en modelado de lenguaje.

En referencia a mejoras del coste computacional, la última tendencia es la utilización de "Graphics Processing Units" (GPUs) junto a una descripción matricial del algoritmo de entrenamiento [Schwenk *et al.*, 2012]. Esto permite reducir enormemente el tiempo necesario para el entrenamiento de los modelos.

2.7 Resumen

Este capítulo ha presentado el estado del arte dentro del ámbito del modelado de lenguaje. Se ha centrado en las técnicas estadísticas más importantes y en la aproximación conexionista, los NN LMs o *modelos conexionistas de lenguaje*.

Las mejores prestaciones se obtienen combinando las redes neuronales con modelos de N-gramas estimados mediante el descuento de Kneser-Ney modificado. Esta es la combinación que hemos seguido, con resultados muy satisfactorios, estadísticamente significativos en la mayoría de los casos estudiados en este trabajo, como veremos más adelante. El capítulo ha finalizado con un resumen sobre los últimos avances en modelado conexionista de lenguaje, dando lugar a posibles trabajos futuros que ampliaremos en el siguiente capítulo.

3

APORTACIONES A LOS NN LMS ESTÁNDAR

Índice

Este capítulo resume las aportaciones de este trabajo al campo del modelado conexionista del lenguaje, extendiendo la arquitectura estándar con nuevas ideas:

- Integración de los NN LMs en el proceso de decodificación de forma totalmente acoplada.
- Reducción del coste computacional en la fase de evaluación de los NN LMs, una de las aportaciones más importantes de este trabajo, que será introducida en este capítulo y detallada en el siguiente.
- Extensión de los NN LMs con un módulo tipo caché [Kuhn & Mori, 1990] que les permite adaptar la distribución de probabilidad al contexto previo de la frase. Esta aportación será introducida en este capítulo y detallada en el capítulo 5.

3.1 Mejorando los NN LMs

La sección 2.5 ha introducido las principales deficiencias que poseen los NN LMs. Expondremos aquí las soluciones propuestas en la literatura con el fin de evitar las deficiencias, así como las ideas aportadas en este trabajo en esa misma línea.

3.1.1. Proyección de las palabras poco frecuentes en el espacio continuo

Mejorar la proyección de las palabras en el espacio continuo permite mejorar la predicción y el cómputo de las probabilidades condicionales con NN LMs. La codificación de las palabras poco frecuentes no se entrena suficientemente para que su proyección sea útil [Hai-Son *et al.*, 2010]. En este trabajo adoptaremos la siguiente solución: las eliminaremos del vocabulario de la entrada del NN LM. Denotaremos con Ω^I al vocabulario de entrada del NN LM, subconjunto del vocabulario Ω completo de la tarea. Añadiremos una unidad extra a la entrada de la red que aglutinará a todas aquellas palabras que se queden fuera del vocabulario Ω^I. Para contrastar esta aproximación, se han realizado experimentos y no se han encontrado diferencias significativas entre dejar la entrada de la red neuronal con el vocabulario completo Ω o el vocabulario reducido Ω^I (véase la sección 3.1.3). Sin embargo, ante resultados idénticos, un vocabulario reducido permite una reducción del número de parámetros en los modelos si bien la diferencia en el coste computacional es despreciable[1].

Otros autores también han realizado un análisis de este problema. En [Hai-Son *et al.*, 2010] se presentan diversas ideas para mejorar esta proyección:

- Por un lado comentan la posibilidad de reentrenar las redes, manteniendo los pesos de la capa de proyección de una red entrenada anteriormente. Este proceso puede ser iterativo, de manera que puede repetirse hasta que se obtenga convergencia.
- Otra de las mejoras que presentan consiste en inicializar la matriz de pesos de la capa de proyección de manera que todas las palabras tengan la misma codificación, elegida de manera aleatoria al comienzo del entrenamiento.

El segundo método es el que mejores resultados obtiene. Logra una mejora de 10 puntos de PPL y de 0.1 puntos de BLEU. Este método funciona mejor porque consigue que las palabras poco frecuentes tengan todas una codificación muy parecida, ya que su baja frecuencia de aparición hace que sean modificadas muy poco por el algoritmo de entrenamiento. Las palabras frecuentes sufren una modificación en su codificación mucho mayor, de manera que el algoritmo de entrenamiento tenderá a aproximar palabras parecidas en el espacio continuo y a separar aquellas que no tienen nada que ver. Es importante destacar que esta aproximación funciona bien porque los autores no utilizan el weight decay en la capa de proyección al entrenar los NN LMs.

En esta tesis sí se aplica el weight decay a la capa de proyección, por lo que esta aproximación no tiene a priori porqué lograr mejores resultados (véase la sección 2.3.2). Se han realizado experimentos preliminares comparando este método (codificación aleatoria común a todas las palabras y sin weight decay) frente al método estándar (inicialización aleatoria) con nuestro algoritmo de entrenamiento que sí utiliza el weight decay en la capa de proyección y no se han encontrado diferencias entre ambas aproximaciones. No obstante se deja como trabajo futuro una investigación más profunda de este punto.

[1] Nótese que tanto en la fase de entrenamiento como en la de evaluación la proyección de las palabras tiene el coste de acceder a una tabla, y por tanto el tamaño del vocabulario no tiene impacto en la eficiencia de los algoritmos.

3.1.2. Mejoras del coste computacional

Los problemas de eficiencia de las redes neuronales son muy similares en la fase de entrenamiento y la fase de utilización o evaluación. El problema subyacente es el mismo, el cálculo asociado a la normalización softmax, pero la solución al problema es diferente, ya que en la fase de evaluación es posible mejorar mucho más el rendimiento de la red neuronal.

En la fase de entrenamiento el coste computacional de la red neuronal es menos importante que en la fase de test. Sin embargo, la escala en tamaño que estos últimos años han sufrido los corpus textuales, incide en un incremento directo del tiempo necesario para el entrenamiento. Se llega hasta tal punto que es poco frecuente poder realizar un entrenamiento completo, entendiendo por completo que la red neuronal haya visitado durante el entrenamiento *todas* las muestras disponibles. Ya se han comentado algunas de las posibles mejoras del entrenamiento, como el usar la variante estocástica del BP. En la literatura se plantean algunas ideas que han sido adoptadas desde el principio en este trabajo.

El uso de 32 bits de precisión en lugar de 64

Los números en coma flotante en la implementación de los modelos es de simple precisión (32 bits). Se ha demostrado en [España-Boquera *et al.*, 2007a; Schwenk, 2007] que es suficiente precisión para entrenar redes neuronales, y permite que los datos ocupen la mitad de espacio. Asimismo, el ahorro en memoria mejora la localidad de los datos en la caché del ordenador, y las operaciones aritmético-lógicas son más sencillas [España-Boquera *et al.*, 2007a]. Esta mejora en la implementación permite que los algoritmos sean 1.5 veces más rápidos [Schwenk, 2007].

Reducir el tamaño de la capa de salida de la red neuronal

Reducir el tamaño de la capa de salida significa decrementar el coste computacional en un factor muy importante. Esta capa es la que domina el coste del paso forward y backward de los NN LMs. Se utilizará la idea de la "Short-List" [Schwenk, 2007] para reducir el tamaño de esta capa, que consiste en limitar la salida de la red a las 10K o 20K palabras más frecuentes. Denotaremos el vocabulario de la Short-List como Ω'. El resto de palabras serán reconocidas como palabras fuera de la Short-List: $OOS = \Omega - \Omega'$.

En la fase de evaluación, el uso de la Short-List obliga a que exista una manera de repartir la probabilidad de las palabras que estén en el conjunto OOS. En la literatura se pueden encontrar diversas propuestas. [Schwenk, 2007] propone utilizar la siguiente ecuación:

$$p(y_j|\bar{h}_j) = \begin{cases} p_{NN}(y_j|\bar{h}_j)p_{NG}(\neg OOS|\bar{h}_j), & \text{sii } y_j \in \Omega' \,; \\ p_{NG}(y_j|\bar{h}_j), & \text{sii } y_j \notin \Omega' \,, \end{cases} \tag{3.1}$$

donde $p_{NG}(\neg OOS|\bar{h}_j) = \sum_{\omega \in \Omega'} p_{NG}(\omega|\bar{h}_j)$ es un factor de normalización que asegura que la suma de la masa de probabilidad es 1, siendo $p_{NG}(\cdot)$ la probabilidad usando un modelo de N-gramas estándar, y $p_{NN}(\cdot)$ la probabilidad usando la red neuronal. También propone una alternativa consistente en extender la salida de la red neuronal con una nueva neurona que aglutine a todas las palabras que están fuera de OOS, quedando de esta manera:

$$p(y_j|\bar{h}_j) = \begin{cases} p_{NN}(y_j|\bar{h}_j), & \text{sii } y_j \in \Omega' \,; \\ p_{NN}(OOS|\bar{h}_j) \cdot p_{NG}(y_j|\bar{h}_j) \cdot p_{NG}(OOS|\bar{h}_j), & \text{sii } y_j \notin \Omega' \,, \end{cases} \tag{3.2}$$

siendo $p_{NG}(OOS|\bar{h}_j) = \sum_{\omega \notin \Omega'} p_{NG}(\omega|\bar{h}_j)$ un factor de normalización para que la masa de probabilidad sume 1.

[Emami & Mangu, 2007] plantean reducir el coste de calcular $p_{NG}(OOS|\bar{h}_j)$ o su complementaria $p_{NG}(\neg OOS|\bar{h}_j)$ en tareas de gran envergadura, evitando que pueda repercutir en un incremento del tiempo (asumiendo que se calcule al vuelo este factor), o bien en un incremento del espacio en memoria (si se precalcula y se guarda junto al N-grama). Proponen la z-norm, que consiste en usar esta ecuación:

$$p(y_j|\bar{h}_j) = \begin{cases} p_{NN}(y_j|\bar{h}_j), & \text{sii } y_j \in \Omega' ; \\ 0, & \text{sii } y_j \notin \Omega' , \end{cases} \tag{3.3}$$

que necesariamente debe ser combinada de forma lineal con un N-grama estadístico estándar para funcionar, y da resultados equivalentes a la normalización presentada en la ecuación (3.2). Sigue asegurando que la suma de la masa de probabilidad es 1 ya que la red neuronal no dispone de la salida para palabras del conjunto OOS, y por tanto la masa de probabilidad se reparte únicamente entre las palabras que pertenecen a Ω'.

En [Park *et al.*, 2010] utilizan otra modificación de la ecuación. Añaden a la red neuronal una neurona de salida para el conjunto OOS, pero asumen que la masa de probabilidad que la red neuronal da a ese conjunto de palabras es idéntica al factor $P_{NG}(OOS|\bar{h}_j)$. La ecuación queda así:

$$p(y_j|\bar{h}_j) = \begin{cases} p_{NN}(y_j|\bar{h}_j), & \text{sii } y_j \in \Omega' ; \\ p_{NG}(y_j|\bar{h}_j), & \text{sii } y_j \notin \Omega' , \end{cases} \tag{3.4}$$

asumiendo como hemos dicho que $p_{NN}(OOS|\bar{h}_j) = p_{NG}(OOS|\bar{h}_j)$. Sin embargo, esto no tiene porque ser así, y consecuentemente no es posible asegurar que la masa de probabilidad sume 1, y por tanto el modelo no está normalizado.

En esta tesis se ha optado por una aproximación de compromiso a todas estas. Añadimos a la red neuronal una salida que permite aglutinar todas las palabras que pertenecen al conjunto OOS. Esta probabilidad no se puede utilizar directamente en la fase de evaluación, ya que se trata de una estimación de la probabilidad de la suma del conjunto completo, no de una palabra del conjunto. La solución de compromiso consiste en utilizar un unigrama estimado sobre el conjunto OOS con los datos de entrenamiento. Por tanto, es una especialización de la ecuación (3.2) para el caso en que $p_{NG}(\cdot)$ es un unigrama:

$$p(y_j|\bar{h}_j) = \begin{cases} p_{NN}(y_j|\bar{h}_j), & \text{sii } y_j \in \Omega' ; \\ p_{NN}(OOS|\bar{h}_j) \cdot \dfrac{\mathcal{C}_{OOS}(y_j)}{\sum\limits_{y' \notin \Omega'} \mathcal{C}_{OOS}(y')}, & \text{sii } y_j \notin \Omega' , \end{cases} \tag{3.5}$$

donde la probabilidad calculada por la red neuronal, $p_{NN}(\cdot)$, es:

- la probabilidad dada por la red neuronal cuando y_j está en el vocabulario de la Short-List Ω';
- o bien si está fuera del vocabulario Ω' se suaviza la probabilidad de la neurona OOS usando el unigrama calculado mediante las cuentas de las palabras que están fuera de la Short-List (conjunto OOS).

Normalmente el tamaño de la Short-List puede variar entre las 10K palabras y las 20K palabras más frecuentes del vocabulario completo de la tarea. La ley de Zipf [Zipf, 1949] establece que la frecuencia de cualquier palabra es inversamente proporcional a su posición en la tabla de frecuencias. Esto permite que una Short-List de este tamaño pueda cubrir entre el $85\,\%$ y el $95\,\%$ de las palabras del conjunto de test, en función de la tarea. La mejora computacional obtenida mediante esta idea es lineal con el tamaño del conjunto OOS y proporcional al valor $|\bar{H}||OOS|$ siendo $|\bar{H}|$ la talla de la capa oculta del NN LM.

Entrenamiento en modo bunch

Otra mejora de la eficiencia del entrenamiento de las redes neuronales que ha sido utilizada en este trabajo es el "bunch mode". Consiste en modificar el algoritmo para que en lugar de ser on-line, funcione en modo "batch", pero para un número de muestras de entrenamiento muy reducido. Se agrupan las muestras de una época en bloques de 32, o 64, y se ejecuta un "forward" con todas ellas a la vez. Los pesos se modifican con la información de todo el bunch, es decir, con la información del error medio generado por todas las muestras del bloque. Esto permite que las multiplicaciones matriz por vector del algoritmo original se conviertan en multiplicaciones matriz por matriz. La arquitectura hardware actual permite una implementación muy eficiente de estas operaciones. Se utiliza para ello el interfaz de la biblioteca CBLAS, que ofrece operaciones matemáticas en C con matrices y vectores siguiendo el estándar BLAS [Lawson *et al.*, 1979; BLAS, 1979]. Dicho interfaz se puede compilar contra varias bibliotecas de cálculo numérico que incorporan implementaciones muy eficientes de las operaciones matriciales necesarias. Esta mejora permite multiplicar por 10 la velocidad de los algoritmos implementados [Schwenk, 2007; Palacios, 2012].

Aproximaciones útiles en la fase de evaluación

Además de las ideas anteriores, que se pueden aplicar en las dos fases, la de entrenamiento y la de evaluación, hay más aproximaciones que se pueden utilizar en la fase de evaluación y permiten obtener mejoras en la eficiencia de los modelos:

- Es posible descomponer el cómputo de la probabilidad de forma jerárquica, obteniendo un árbol binario de decisión y probabilístico, que permite reemplazar el coste $O(N)$ de la capa de salida por $O(\log N)$ [Morin & Bengio, 2005; Hai-Son *et al.*, 2011]. Esta aproximación sirve para acelerar tanto el entrenamiento como la evaluación, si bien está en este apartado porque su repercursión en evaluación es más importante para esta tesis.

- Como aportación de este trabajo se presenta una idea novedosa en esta línea. Consiste en precalcular y almacenar las constantes de normalización de la activación softmax más importantes, de manera que muy probablemente se utilicen con mucha frecuencia durante la etapa de evaluación. De esa manera no es necesario calcular *toda* la capa de salida para obtener la probabilidad de una única palabra. Esta aportación se describe con mayor profundidad en el capítulo 4.

3.1.3. Experimentos con distintos tamaños de vocabulario de entrada

Para ilustrar el efecto del tamaño del vocabulario Ω^I en la entrada del NN LM, presentamos experimentos donde se varía el tamaño de dicho vocabulario para distintas tareas. Los

detalles de la experimentación se encuentran en los capítulos 8 y 11, aquí sólo vamos a recoger los resultados más destacables haciendo rescoring de listas de N-best con los NN LMs. La técnica usada para la Short-List sigue la ecuación (3.5), y además todos los NN LMs son la combinación lineal de cuatro redes neuronales. Para la tarea IAM-DB, los NN LMs han sido combinados linealmente con un bigrama estándar, y los resultados de PPL y WER se corresponden a dicha combinación. En la tarea BTEC el NN LM se combina log-linealmente con un N-grama estándar, el resultado de PPL se corresponde al NN LM sin combinar log-linealmente, mientras que el resultado en BLEU sí que incorpora dicha combinación. La aplicación de la ecuación (3.5) permite asegurar que los resultados son comparables con un N-grama estándar, ya que el vocabulario final del modelo es el vocabulario completo de la tarea.

La tabla 3.1 muestra resultados en dos tareas diferentes, para distintos valores de N en los NN LMs, y para distintos tamaños de vocabulario de entrada Ω^I (formado por aquellas palabras con mayor frecuencia de aparición en el conjunto de entrenamiento).

- Tarea de reconocimiento de escritura manuscrita IAM-DB: tarea de tamaño medio, donde el vocabulario de la partición de entrenamiento para modelos de lenguaje asciende a 103K palabras. Se han probado cuatro tamaños distintos de Ω^I, usando en todos los casos una Short-List $\Omega' = 10$K palabras en la salida de las redes neuronales. Se puede observar que el aumento del tamaño del vocabulario no aporta diferencias significativas ni en PPL ni en WER. No obstante, sí que es significativa la diferencia en WER comparando el modelo de N-gramas estándar y los NN LMs.

- Tarea de traducción automática BTEC'06 Italiano-Inglés: tarea de tamaño reducido, con un vocabulario en el entrenamiento de 7.2K palabras. Se han probado tres configuraciones de tamaño para Ω^I. En este caso no se observan diferencias significativas en PPL, pero sí que se observan en BLEU. La diferencia con el N-grama estándar muestra una ventaja superior en los NN LMs. La diferencia entre ambos es mayor que para el caso anterior, ya que los NN LMs permiten sacar mucho partido a tareas donde los datos para entrenar son reducidos.

En la tarea IAM-DB no se observan diferencias significativas entre las distintas tallas de vocabulario de entrada, ni a nivel de WER ni de PPL. Para el caso de la tarea BTEC, donde el tamaño de la capa de salida es más parecido al tamaño del vocabulario de la tarea, sí que se aprecian diferencias importantes al aumentar la talla del vocabulario.

A partir de estos resultados, concluimos que usar un vocabulario reducido en la entrada puede ser igual de útil que usar uno de mayor tamaño, al menos para los conjuntos de test utilizados. Sin embargo, desde el punto de vista teórico es obvio que hay un gran margen de mejora en este aspecto. En entornos de producción y uso real de la aplicación, un vocabulario más grande en la entrada puede aportar una mayor cobertura, siendo más fiable.

3.2 Incorporación de NN LMs en sistemas automáticos de transcripción

En los sistemas de reconocimiento automático del habla/escritura y de traducción automática estadística, el modelo de lenguaje forma parte integrante de todo el sistema a través de las ecuaciones (1.2) o (1.3). Una forma compacta y eficiente de representar los modelos de

Tabla 3.1: *Resultados en tareas de reconocimiento de escritura manuscrita y traducción automática usando NN LMs con distinto tamaño de vocabulario de entrada Ω^I. //* Results using different input vocabulary configuration for NN LMs on HTR and SMT tasks. Referenced at page 299.

IAM-DB task, Validation set

System	$\lvert\Omega^I\rvert$–$\lvert\Omega'\rvert$	PPL	% WER
2-gram standard	103K–103K	659	17.3
NN LM-2gr	10K–10K	527	16.0
NN LM-3gr	10K–10K	496	15.8
NN LM-4gr	10K–10K	479	15.8
NN LM-2gr	17K–10K	535	15.9
NN LM-3gr	17K–10K	497	15.9
NN LM-4gr	17K–10K	491	15.9
NN LM-2gr	19K–10K	543	16.0
NN LM-3gr	19K–10K	503	15.8
NN LM-4gr	19K–10K	493	15.8
NN LM-2gr	56K–10K	538	16.0
NN LM-3gr	56K–10K	505	15.9
NN LM-4gr	56K–10K	497	15.7

Italian-English BTEC task, Dev set

System	$\lvert\Omega^I\rvert$–$\lvert\Omega'\rvert$	PPL	% BLEU
4-gram standard	7.2K–7.2K	155	39.2
NN LM-3gr	2.4K–2.4K	131	41.7
NN LM-4gr	2.4K–2.4K	130	40.9
NN LM-3gr	3.1K–3.1K	132	41.6
NN LM-4gr	3.1K–3.1K	128	42.0
NN LM-3gr	4.1K–4.1K	130	41.8
NN LM-4gr	4.1K–4.1K	130	42.6

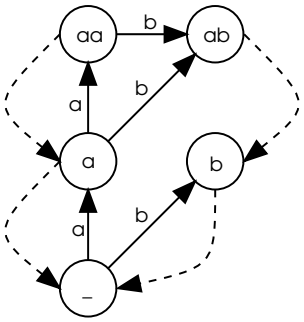


Figura 3.1: *Ejemplo de modelo de lenguaje de N-gramas (3-grama) representado en forma de autómata. Las aristas con línea discontinua son las que hacen referencia al suavizado por back-off. // N-gram language model as a finite state automaton. Referenced at page 299.*

lenguaje de N-gramas es como autómatas finitos estocásticos [Llorens Piñana, 2000], permitiendo representar tanto modelos de lenguaje interpolados como suavizados por back-off. De esta manera, el algoritmo de Viterbi [Viterbi, 1967] implementado para encontrar la secuencia de palabras que maximiza la probabilidad tan sólo necesita un identificador entero del estado del autómata que representa al contexto $\bar{h}_j$ de la siguiente hipótesis. Dicho identificador lo denominaremos LMkey, y se va actualizando conforme el sistema va procesando hipótesis. Un LMkey representa a una secuencia de $N-1$ palabras, es decir, un $N-1$-grama. Todas las hipótesis tienen un LMkey, y cuando la hipótesis se extiende con una nueva palabra, se transita en el autómata con dicha palabra y el estado destino pasa a ser el nuevo LMkey. Este procedimiento permite que la búsqueda en el modelo de lenguaje sea muy eficiente. La figura 3.1 muestra un ejemplo sencillo de representación de un modelo de lenguaje de N-gramas como autómata.

3.2.1. Generalización de los modelos de lenguaje

Se han generalizado los algoritmos relacionados con modelos de lenguaje con el objetivo de permitir el uso de NN LMs que, en realidad, son una combinación lineal o log-lineal de distintos NN LMs y N-gramas (como se ha explicado en la sección 2.4), o bien el uso de modelos de lenguaje de N-gramas estándar. Para llevar a cabo esta generalización se ha definido una clase abstracta C++ que implementa el interfaz de la tabla 3.2.

3.2.2. NN LMs como modelos de estados finitos

Los NN LMs también se pueden representar como autómatas finitos estocásticos, dado que siguen siendo modelos de lenguaje de N-gramas. Pero, como ya se ha comentado, representan al autómata completo que permite calcular la probabilidad de todos los posibles N-gramas. Esto quiere decir que representa al conjunto de todos los $|\Omega|^N$ N-gramas diferentes para un vocabulario Ω. No es posible por tanto expandir el modelo completo, por su enorme tamaño. Sin embargo, sí que es posible generar dicho modelo *a demanda*: Al principio, el autómata sólo contiene el estado inicial, y el resto de estados se van enumerando conforme llegan hipótesis nuevas, y con ellas N-gramas diferentes de los vistos. Cada estado del autómata se identifica con una secuencia de $N-1$ palabras, y tiene el mismo significado

Tabla 3.2: *Interfaz genérico para modelos de lenguaje. //* Generic interface for language models. Referenced at page 299.

prepareLM(LMKey)	Prepares the LM for an LMkey. If it is an NN LM, this method executes the forward algorithm of the neural network, if it is a standard N-gram, this method is void.
score getLMprob(LMKey, word)	Returns the LM probability, given the LMkey and the **word**.
LMKey getNextLMkey(LMKey, word)	Walks using a transition in the LM from the given LMkey and the given **word**.
LMKey getInitialLMkey()	Returns the initial LMkey which represents the context cue *bcc*.
LMKey getFinalLMkey()	Returns the final LMkey which represents the context cue *ecc*.
restartLM()	Reinitializes the LM. In NN LMs this method erases the TrieLM dynamic part, and in case of standard N-grams this method is void.

que el LMkey. Para implementar este procedimiento, los NN LMs incorporan una estructura de datos tipo Trie que denominaremos TrieLM y que permite representar de forma compacta secuencias de N-gramas, de manera que un mismo TrieLM puede contener estados para NN LMs de diferente orden N. Esta estructura de datos contiene dos partes, una persistente, y otra dinámica:

- la parte persistente permite tener algunos N-gramas especiales ya introducidos en el TrieLM, de manera que *nunca* cambian su identificador (son estados del autómata ya enumerados);
- la parte dinámica permite crear a demanda nuevos identificadores para N-gramas, que serán borrados cuando termine la decodificación de una frase, liberando así el espacio ocupado.

El TrieLM se implementa como una tabla Hash con direccionamiento abierto sobre un vector de tamaño estático. Esto posibilita que cada nodo pueda identificarse por la posición que ocupa en dicho vector, ya que no se le permite crecer en tamaño. La figura 3.2 ilustra un ejemplo de esta estructura de datos. Cada nodo del TrieLM contiene:

- Índice al padre: valor entero que indica la posición, dentro del vector, que ocupa el padre del nodo actual.
- "Timestamp": permite controlar *cuándo* fue creado el nodo. En el TrieLM existe un contador que cada vez que se borra la parte dinámica se incrementa en uno. Hay un valor especial, el cero, que identifica a los nodos persistentes, que nunca son borrados. Borrar la parte dinámica del TrieLM consiste en incrementar el valor del contador global que lleva la cuenta del timestamp actual, de manera que siempre que se busca un nodo se comprueba primero si su timestamp es diferente del valor actual del contador del TrieLM; si es así, se asume que dicho nodo ha sido borrado y, por tanto, está vacío.

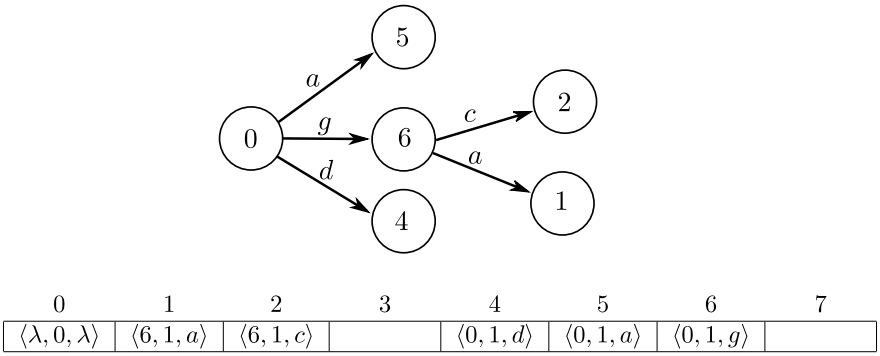


Figura 3.2: *Ejemplo de TrieLM con vector de tamaño máximo* 8*. Cada tupla* $\langle x, y, z\rangle$ *tiene tres elementos, siendo* x *el índice del nodo padre,* y *el timestamp,* y z *la palabra de transición. El TrieLM ha sido creado a partir de este conjunto de* N*-gramas:* $\{a, gc, ga, d\}$*. //* TrieLM of size 8. Each $\langle x, y, z\rangle$ tuple is: parent index, timestamp, and transition word. Referenced at page 300.

- Word: es un índice numérico que identifica a una palabra del vocabulario Ω.
- Dado un nodo y una palabra, se calcula una huella o "hash" de dicho par, y se busca en una tabla con direccionamiento abierto, de esa manera sabemos cual es el nodo destino. En caso de no encontrar nodo destino, es posible añadirlo en ese momento y devolverlo.

El TrieLM implementa el interfaz descrito en la tabla 3.3.

3.2.3. NN LM integrado durante el proceso de búsqueda

A partir de la estructura de datos TrieLM de los NN LMs, es posible implementar un algoritmo de Viterbi genérico que, de forma independiente al modelo de lenguaje, implementa la búsqueda del mejor camino. El estado del modelo de lenguaje siempre será un número entero, siendo este un estado del autómata del modelo de N-gramas estándar, o bien un nodo del vector del TrieLM. De esta manera integrar NN LMs en la búsqueda se convierte en algo trivial. Sin embargo, los problemas computacionales que aparecen son enormes. Como ya se ha comentado, en el capítulo 4 se explica la solución propuesta en este trabajo a los problemas computacionales de los NN LMs.

3.2.4. Repuntuando listas de N-best

Una forma de evitar los problemas de eficiencia de los NN LMs es desacoplar su uso, tal y como aparece siempre en toda la literatura. De esta manera tenemos dos etapas:

1. Etapa de reconocimiento/traducción: el proceso de búsqueda genera una lista (o bien un grafo) de las N-best hipótesis.
2. Etapa de rescoring de la lista de N-best (o grafo): el NN LM se utiliza para puntuar cada hipótesis generada en la etapa anterior. Esta nueva puntuación se combina con las puntuaciones de la etapa 1 en un nuevo proceso de maximización, y se extrae así la

Tabla 3.3: *Interfaz para el objeto TrieLM. //* TrieLM interface. Referenced at page 300.

`node getParent(node)`	Returns the parent of a **`node`**.
`node getWord(node)`	Gets the word for the incoming transition to the given **`node`**.
`bool hasChild(node, word, &dest)`	Searchs an outgoing transition with **`word`** in the given **`node`**. If it exists, the method returns **`True`** and keeps in **`dest`** the transition destination. Otherwise, the method returns **`False`**.
`node getPersistentChild(node, word)`	Returns the destination node after walk with **`word`** from **`node`**. If the transition doesn't exists, it will be created as a persistent transition. This method is allowed only if the dynamic part of the TrieLM is empty.
`node getChild(node, word)`	Is similar to the previous method, but creating a dynamic transition in case of no existence.
`vector getSequence(node)`	Returns the path of words that walks from the TrieLM root node to the given **`node`**.
`node searchSequence(vector)`	Returns the destination node after walk from the root node with the given words path **`vector`**. It creates dynamic transitions if needed.
`node rootNode()`	Returns the TrieLM root node identifier.
`clear()`	Erases the TrieLM dynamic transitions.

Algoritmo 3.1 Calcula la probabilidad del NN LM para la lista de las N-best hipótesis de una frase. *// NN LM probability computation for an N-best list. Referenced at page 300.*

Require: A list $\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_K$ of the K-best hypotheses extracted from the searching procedure of one sentence.[2]

Ensure: Computes the probability of the NN LM for each of the K hypotheses in a probability vector $Prob$

1: **function** rescoringNbest($\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_K$)
2: **begin**
3: Let $Hash$ a hash table that associates to each possible $N-1$-gram (LMkey) a list of pairs $\langle$next word, K-best hypothesis index$\rangle$
4: **for** $i = 1, \ldots, K$ **do**
5: Let $s =$ searchSequence(bcc^{N-1}) the initial NN LM input formed by $N-1$ begin context cues // s is a TrieLM node identifier (LMkey)
6: **for all** word $\omega \in \bar{y}_i$ **do**
7: Append pair $\langle \omega, i \rangle$ to the list associated with LMkey s at $Hash$
8: Let $\bar{v} =$ getSequence(s) the word sequence corresponding to TrieLM node s
9: Update $s =$ searchSequence($v_2 v_3 \ldots v_{N-1} \omega$), the next TrieLM node identifier corresponding to the next $N-1$-gram
10: **end for**
11: Append pair $\langle ecc, i \rangle$ to the list associated with LMkey s at $Hash$
// Auxiliar vector to compute the sentence probabilities
12: **end for**
13: Let $Prob$ an all-ones vector of size K that stores the probability computed by the NN LM for each of the K-best hypotheses
14: **for all** TrieLM node LMkey $s \in Hash$ **do**
15: Let $\bar{O} =$ forward(getSequence(s)), the NN LM output for input $N-1$-gram s
16: **for all** pair $\langle \omega, i \rangle$ associated with LMkey s at $Hash$ **do**
17: Update $Prob(i) = Prob(i) \cdot o_\omega$, being o_ω the NN LM probability $p(\omega|s)$
18: **end for**
19: **end for**
20: **return** $Prob$
21: **end function**

mejor hipótesis. Esta etapa tiene dos fases, la fase de optimización de los pesos de la combinación y la de test:

- Durante la optimización se vuelve a repetir el MERT sobre las listas utilizando los scores del NN LM.
- En la fase de test se usan los pesos ya optimizados.

El algoritmo 3.1 calcula la puntuación del NN LM para cada frase de la lista de N-best. La figura 3.3 ilustra el funcionamiento del rescoring en sus dos fases, la de optimización de pesos y la fase de test. Nótese que, en la fase de test, la segunda búsqueda que aparece en el diagrama no utiliza el sistema completo, es una búsqueda sobre la lista de N-best utilizando los pesos óptimos en la combinación log-lineal. El algoritmo hace uso de dos tablas:

- $Hash$: una tabla Hash indexada por el nodo del TrieLM guarda una lista de pares. Cada par está compuesto por una palabra y el índice de la frase en la lista de N-best que originó esta entrada en la tabla;
- $Prob$: un vector donde ir acumulando la probabilidad del NN LM para cada hipótesis de la lista.

Estas tablas permiten implementar un algoritmo muy eficiente que aglutina todos los $N-1$-gramas de la lista para únicamente lanzar un forward de la red neuronal del NN LM por cada posible $N-1$-grama, y así disponer en la capa de salida de la probabilidad de todas las palabras.

La adaptación del algoritmo 3.1 para que funcione con grafos de palabras requiere un poco más de cuidado. Habría que calcular el producto cartesiano entre el NN LM y el grafo de palabras de entrada. Dicho producto cartesiano se puede calcular de manera que todos los $N-1$-grama iguales sean calculados en un único forward del NN LM.

Como ya se comentó en la sección 2.4, dependiendo de la tarea, la probabilidad del modelo de N-gramas estándar puede formar parte de la combinación log-lineal cuando se introduce el NN LM (dando lugar a dos modelos de lenguaje en dicha combinación), o bien se combinan linealmente ambos modelos (utilizando un único modelo de lenguaje en la combinación log-lineal).

3.3 Propuestas de mejora

3.3.1. Adaptación al contexto con NN LMs tipo caché

El origen de esta extensión se inspira en los modelos de lenguaje con caché [Kuhn & Mori, 1990]. La idea básica consiste en ampliar la entrada del NN LM para que pueda modelar el contexto de las frases reconocidas previamente. A través de esta extensión el NN LM adapta al contexto la distribución de probabilidad que calcula en su salida. Los detalles del trabajo realizado para esta tesis se encuentran en el capítulo 5.

3.3.2. Información sintáctica para inicializar la capa de proyección

Ya hemos comentado en el comienzo del capítulo que la proyección de las palabras poco frecuentes plantea un problema en los NN LMs. La propuesta que aquí hacemos (dejando su

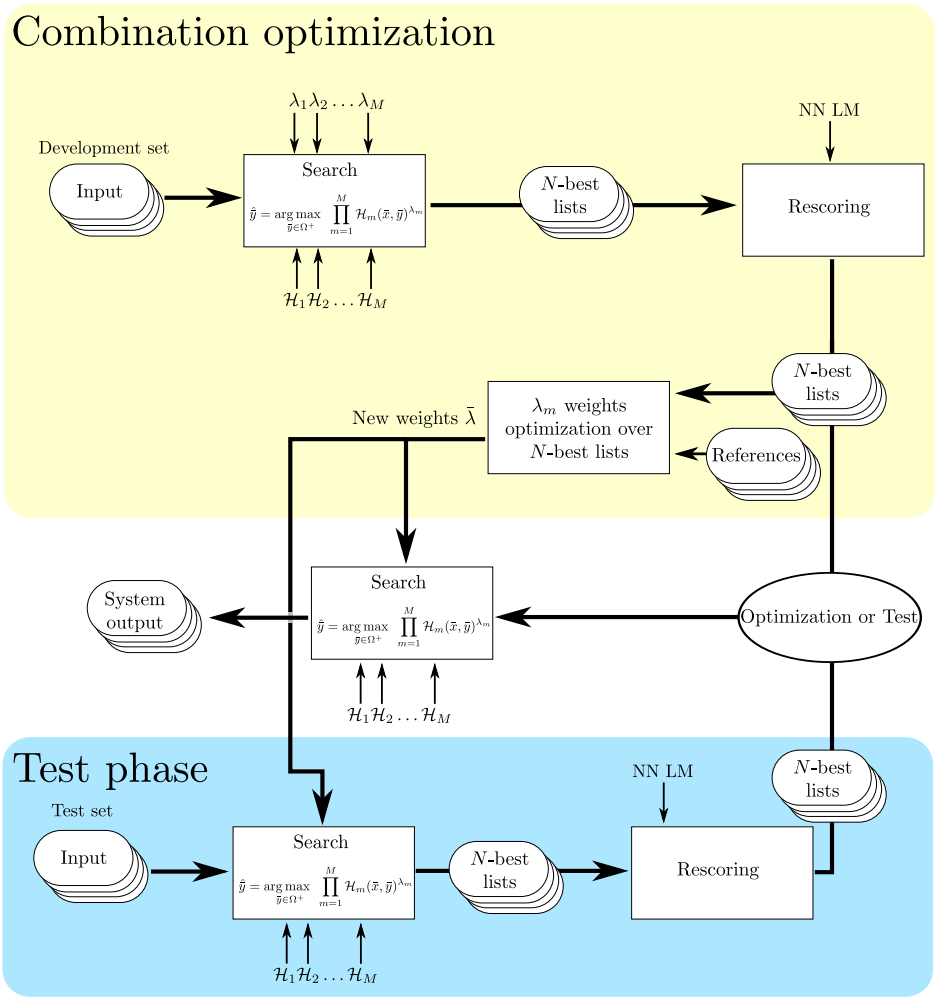


Figura 3.3: *Proceso de repuntuación de listas de N-best mediante NN LMs y sus dos fases: fase de optimización de pesos y fase de test. //* N-best list rescoring process using NN LMs. Referenced at page 300.

experimentación como trabajo futuro) consiste en utilizar información sintáctica para inicializar los pesos de dicha capa. El objetivo es inicializar la capa de proyección de manera que la proyección inicial de las palabras esté en función de las etiquetas "Part-Of-Speech" (POS) que puede tener cada una de ellas, teniendo en cuenta la frecuencia relativa de cada etiqueta en cada palabra. El método constaría de estos pasos:

1. Seleccionar un tamaño P para la proyección de cada palabra.
2. Por cada etiqueta POS crear un vector aleatorio de tamaño P, que denotaremos como $\bar{p}_t$.
3. Necesitamos tener precalculada la $p(t|\omega)$ para toda posible etiqueta POS t.[3] Asignamos a la palabra ω una codificación calculada como sigue:

$$\bar{p}_\omega = \sum_t p(t|\omega)\bar{p}_t$$

   donde $\bar{p}_\omega$ es la codificación inicial para la palabra ω.
4. Finalmente, podemos perturbar con ruido Gaussiano, de media 0 y desviación típica pequeña, la codificación inicial de cada posible palabra ω del vocabulario Ω, añadiendo variabilidad para ayudar la convergencia del algoritmo de entrenamiento.

De esta manera permitimos que aquellas palabras que comparten etiquetas POS tengan una inicialización muy parecida. Así las palabras poco frecuentes se encontrarán más o menos cerca de otras cuya función sintáctica sea similar.

3.3.3. Inclusión del parámetro momentum

El cálculo estándar del modo bunch para NN LMs, presentado en [Schwenk, 2007], permite acelerar mucho el entrenamiento de los modelos. No obstante dicho cálculo carece de un parámetro que consideramos muy interesante para su uso futuro, como es el "momentum".

Es posible introducir este parámetro en el cómputo utilizando una modificación del algoritmo BP en modo bunch, cuyo coste se incrementa en un factor de $O(2|\mathbf{W}|)$, tal y como muestra el algoritmo 3.2.

El método se especializa para simplificar cálculos en el caso de momentum igual a 0.

3.3.4. Mejoras a la combinación lineal de NN LMs y N-gramas estándar

Los resultados presentados en el reciente trabajo de [Oparin *et al.*, 2012] ofrecen luz sobre nuevas formas de combinar NN LMs y N-gramas estándar con el objetivo de mejorar los resultados de dicha combinación. De este trabajo se desprende un análisis de la PPL en función del orden del N-grama al que el modelo estadístico *baja* para poder computar la probabilidad de un evento. A partir de esos resultados se concluye que:

- Cuando el N-grama estándar calcula la probabilidad utilizando el orden más elevado, su PPL es menor que la de NN LM.

[3] Se puede calcular por conteo. Existen corpus etiquetados que pueden utilizarse para estimar estas probabilidades, o bien puede etiquetarse el corpus de la tarea utilizando un software de etiquetado automático.

Algoritmo 3.2 Actualiza los pesos de una capa de una red neuronal en modo bunch para que acepte momentum en el entrenamiento por BP para NN LMs. // *BP with bunch mode and momentum term. Referenced at page 301.*

Require: A neural network layer described by its weights matrix $\mathbf{W}$, its input neurons matrix $\mathbf{I}$, its output neurons matrix $\mathbf{O}$, its input backprop errors matrix E_I, its output backprop errors matrix E_O, the learning rate η, the momentum μ, and the weight decay ϵ

Ensure: Updates the weights $\mathbf{W}$

1: **function** UpdateWeights($\mathbf{W}$, $\mathbf{I}$, $\mathbf{O}$, $\mathbf{E_o}$, $\mathbf{E_I}$, η, μ, ϵ)
2: **begin**
3: **for** $b = 1, \ldots,$ bunch size **do**
4: **for** $e_i \in E_o(b)$ **do**
5: Update e_i with $e_i \cdot$ derivative of $O(b, i)$
6: **end for**
7: **end for**
8: $\mathbf{E_I} = \mathbf{E_O}\mathbf{W}$ // Compute error backprogation multiplying matrixes with SGEMM CBLAS function.
9: **if** $\mu \neq 0.0$ **then**
10: **for** $w_i \in \mathbf{W}$ **do**
11: $W'(i) = \mu(w_i - w'_i)$
12: **end for**
13: $\mathbf{W}' = (1 - \epsilon)\mathbf{W} + \mathbf{W}'$ // SAXPY CBLAS function.
14: $\mathbf{W}' = -\eta\mathbf{E_O^T}\mathbf{I} + \mathbf{W}'$ // SGEMM CBLAS function.
15: Swap($\mathbf{W}', \mathbf{W}$)
16: **else**
17: $\mathbf{W}' = -\eta{\mathbf{E_O}}^T\mathbf{I} + \epsilon\mathbf{W}'$ // SGEMM CBLAS function.
18: **end if**
19: **end function**

- Cuando el N-grama estándar desciende a órdenes menores para calcular la probabilidad, el NN LM obtiene una PPL menor que la del N-grama estándar, y esta diferencia se hace mayor conforme más órdenes tenga que descender el N-grama estándar.

En esta tesis proponemos dos formas de sacar partido de este análisis para mejorar la forma de combinar los modelos conexionistas de lenguaje:

1. Utilizando el "Generalized Linear Interpolation" (GLI) de [Hsu, 2007], donde se observan mejoras al generalizar la combinación lineal de modelos de lenguaje para que sea dependiente del contexto $\bar{h}_j$ del N-grama.

2. Entrenar un NN LM *obligándole* a estimar las probabilidades del modelo estadístico estándar en aquellos N-gramas donde no es necesario bajar a modelos de orden inferior.

Interpolación lineal generalizada en el espacio continuo

Utilizar GLI [Hsu, 2007] permite mejorar la forma en que se combinan los modelos de lenguaje. GLI puede mejorarse si se utiliza la capa de proyección de los NN LMs, de manera que sea posible estimar los coeficientes de la combinación lineal en función de la proyección de las palabras en el espacio continuo. Veamos la fórmula general que sigue la aproximación de [Hsu, 2007]:

$$p(y_j|\bar{h}_j) = \sum_{r=0}^{R} \alpha(r|\bar{h}_j) p_r(y_j|\bar{h}_j)\,, \tag{3.6}$$

donde $\alpha(r|\bar{h}_j)$ es el peso del modelo r para el contexto $\bar{h}_j$.

En [Hsu, 2007] proponen aprender un modelo basado en una extracción de características a partir de las palabras de $\bar{h}_j$, como pueden ser las cuenta del N-grama $\bar{h}_j$, entre otras. Aquí proponemos utilizar las ventajas de la proyección en el espacio continuo de los NN LMs. De esa forma el proceso de extracción de características es sustituido por una capa de proyección, pudiendo reutilizar la misma que tenemos en los modelos de lenguaje, en caso de disponer alguno ya entrenado. Una red neuronal nos permite estimar los valores de $\alpha_r(r|\bar{h}_j)$, utilizando como función de error la entropía cruzada (o "cross-entropy") del modelo interpolado como función objetivo (véanse los "Mixture Density Networks" de [Bishop, 2006] para entender cómo entrenar la red neuronal de esa forma). La entrada de la red neuronal sería la secuencia de palabras $\bar{h}_j$ y la salida tendría tantas neuronas con activación softmax como modelos a combinar. Es posible inicializar los pesos de la red para que se aproximen al resultado óptimo obtenido para interpolación lineal (LI), de manera que pondríamos en los pesos bias de las neuronas de la salida el logaritmo del coeficiente óptimo para LI. El resto de pesos de la red neuronal se inicializarían con valores muy cercanos a cero, con la intención de perturbar poco la activación de los bias, pero permitiendo que la red no se estanque por tener pesos inicializados a cero.

El entrenamiento de los modelos GLI usando redes neuronales es problemática debido a los datos de entrenamiento. No es posible utilizar la partición de entrenamiento porque en ella el modelo estándar de N-gramas nunca necesita bajar de orden y por tanto mejora a los NN LMs. Estas son las posibilidades que planteamos en este trabajo:

- utilizar una partición de validación, que debe ser suficientemente grande para poder estimar todos los parámetros del modelo;
- usar estrategias que combinen una partición de validación con la incorporación de textos generados de forma aleatoria (posiblemente siguiendo algún tipo de modelo de lenguaje como distribución de probabilidad para generar frases *con cierto sentido*);
- otra opción es utilizar algún tipo de validación cruzada que permita dividir la partición de entrenamiento para entrenar la red neuronal y los modelos de lenguaje estadísticos en particiones distintas;
- finalmente, se puede integrar en un mismo entrenamiento la estimación de los coeficientes de GLI y el propio NN LM, quedando abierto el problema de integrar el entrenamiento de NN LMs y de modelos de N-gramas estadísticos estándar.

Experimento preliminar En la gráfica 3.4 se observan resultados preliminares al extender la GLI con las ideas de la proyección de las palabras en el espacio continuo (GLI-CS). Se trata de un sistema que utiliza un perceptrón (modelo conexionista sin capa oculta) que recibe a su entrada tres palabras codificadas en el espacio continuo y produce a su salida los coeficientes de interpolación lineal de 5 modelos, cuatro NN LMs y un N-grama estadístico estándar. Se ha utilizado la partición de validación de la IAM-DB para entrenar el modelo GLI-CS, y la partición de test para evaluar la capacidad de generalización del modelo. No es una evaluación justa, ya que no tenemos conjunto de validación, y usamos el test a tal efecto, pero no obstante nos sirve para ilustrar las posibilidades futuras de la aproximación. En la gráfica se ha representado en cada época de entrenamiento el valor de PPL obtenida para los siguientes modelos:

- Test LI: PPL obtenida en la partición de test utilizando los coeficientes de LI óptimos para el conjunto de validación.
- Test optimum LI: PPL obtenida en la partición de test utilizando los coeficientes de LI óptimos para el mismo conjunto de test.
- Val optimum LI: PPL obtenida en la partición de validación utilizando los coeficientes de LI óptimos para esa misma partición.
- Test GLI-CS: PPL obtenida en la partición de test usando el modelo neuronal para hacer GLI, optimizando el modelo para el conjunto de validación.
- Val GLI-CS: PPL obtenida en la partición de validación usando el modelo neuronal para hacer GLI, optimizando el modelo para el conjunto de validación.

Es fácil observar como la GLI-CS ha sido capaz de mejorar la PPL obtenida por el modelo Test LI optimum, que sería *la cota superior* utilizando interpolación lineal. [4] Hay que tener en cuenta que este experimento es cerrado y aunque las mejoras obtenidas son pequeñas, es una línea a seguir para mejorar la combinación de NN LMs y N-gramas estándar.

[4] Los resultados de PPL no son comparables con otros mostrados en esta tesis. En este caso no estamos usando Fast NN LMs, y no estamos aplicando las mejoras para reconocer líneas que se comentan en la sección 8.6.

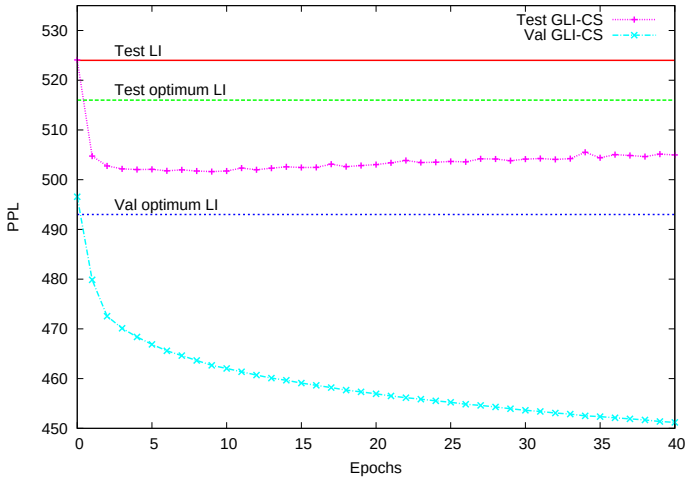


Figura 3.4: *PPL frente al número de épocas de entrenamiento para la interpolación lineal (LI) de modelos de lenguaje y la interpolación lineal generalizada en el espacio continuo (GLI-CS) en la tarea IAM-DB. //* PPL vs training iterations comparing LI and GLI-CS on the IAM-DB task. Referenced at page 301.

3.4 Resumen

Este capítulo ha descrito las aportaciones más importantes de esta tesis al campo del modelado conexionista del lenguaje:

- Se han formalizado los modelos conexionistas de lenguaje para permitir su incorporación *totalmente* integrada en un sistema de decodificación basado en el algoritmo de Viterbi.
- La reducción del coste computacional en la fase de evaluación de los NN LMs es una de las aportaciones más importantes de este trabajo. Esta técnica será expuesta en detalle en el capítulo siguiente.
- Se ha introducido la posibilidad de extender los NN LMs con una entrada tipo caché que permite su adaptación al contexto. Los detalles se encuentran en el capítulo 5.

Se han planteado tres líneas de trabajo futuro:

- Un método que permite inicializar la proyección de las palabras en el espacio continuo a partir de información sintáctica.
- La inclusión del momentum en el algoritmo de aprendizaje.
- El aprendizaje de los pesos de la combinación lineal generalizada (GLI) de modelos de lenguaje mediante una red neuronal que recibe como características la proyección de las palabras en un espacio continuo.

4

EVALUACIÓN RÁPIDA DE NN LMS

Índice

Este capítulo describe el algoritmo utilizado para acelerar la evaluación de los NN LMs, convirtiendo los modelos en lo que denominaremos "Fast NN LMs". Comienza dando los motivos que nos llevan a acelerar dicha evaluación para, más tarde, exponer el desarrollo del algoritmo. Finalmente, se mostrarán resultados experimentales de las técnicas propuestas en diferentes tareas que nos permitirán cuantificar la mejora obtenida así como la calidad del modelo resultante.

Los objetivos y aportaciones de este capítulo son:

- Formalización de la técnica de aceleración del cálculo de la función de activación softmax.
- Comparación de la técnica de aceleración con la aproximación estándar, tanto a nivel de calidad de los modelos como de coste computacional.
- Utilización de NN LMs integrados durante la decodificación en una tarea de reconocimiento de secuencias.
- Comparación de dichos resultados con la aproximación que usa N-gramas estadísticos estándar, evaluando la calidad de los resultados y el coste computacional.

4.1 Motivación

A pesar de las ventajas teóricas de los NN LMs, existe un gran número de dificultades cuando se intentan poner en práctica. El tiempo de cómputo consumido por la evaluación de estos modelos hace que su integración durante el proceso de decodificación tenga un coste demasiado alto. Durante la búsqueda de la mejor hipótesis, el sistema de reconocimiento/traducción genera una cantidad enorme de hipótesis erróneas que necesitan ser evaluadas por el modelo de lenguaje, guiando la búsqueda de la mejor hipótesis hacia aquellas que son lingüísticamente más correctas.

Mayoritariamente, en la literatura sobre NN LMs, estos modelos se utilizan en una segunda etapa, donde se realiza un rescoring de las N-best hipótesis del sistema baseline [Schwenk, 2007, 2010; Mikolov *et al.*, 2010; Hai-Son *et al.*, 2011].

Sin embargo, integrar el NN LM en el proceso de reconocimiento tiene ventajas inherentes muy notables. Se puede extraer *toda* la potencia de los modelos al discriminar las hipótesis buenas de las malas. En un sistema desacoplado tan sólo se permite al modelo conexionista que evalúe las hipótesis que el modelo integrado ha dado por buenas y ha dejado pasar. No obstante, veremos que con un conjunto de las N-best hipótesis lo suficientemente grande ambas aproximaciones se comportan de forma equivalente. En cambio, la integración permite que el sistema sea más sencillo, y que se pueda utilizar en modo on-line, como por ejemplo en sistemas de diálogo, donde la decodificación se realiza al mismo tiempo que el usuario habla.

El objetivo del método de evaluación acelerado que presentamos es la integración *real* de los NN LMs en el proceso de reconocimiento, manteniendo en la medida de lo posible la calidad de los resultados obtenidos.

4.1.1. Acelerando la activación softmax

En modelado del lenguaje conexionista la función de activación para las unidades de la capa de salida es la función softmax [Bishop, 1995], que permite interpretar la salida de la red neuronal como si fueran probabilidades a posteriori. Esta función sigue la ecuación (2.20), que repetimos aquí para facilitar la lectura:

$$o_i = \frac{exp(a_i)}{\sum_{i=1}^{|\bar{A}|} exp(a_i)} \tag{2.20}$$

donde a_i es la activación de la i-ésima unidad de salida, y o_i es el valor de dicha salida tras aplicar softmax. Básicamente se trata de una normalización de las activaciones de las neuronas de salida. El denominador de la ecuación (2.20) es la constante de normalización que limita las salidas de la red neuronal en el intervalo $[0, 1]$, y que permite interpretarlas como probabilidades [Bishop, 1995]. Tiene una ventaja adicional frente a otras funciones de activación que también aseguran, en el límite, tener probabilidades en la salida (como la sigmoide): la convergencia del modelo es mucho más rápida.

Con todo esto, la función de activación softmax es la candidata perfecta para establecer los valores de salida de una ANN. Pero también es la culpable del mayor problema de estos modelos. La constante de normalización softmax obliga a calcular *todas* las salidas de la red, aunque tan sólo se vayan a utilizar unas pocas. Conforme crece el vocabulario, este cómputo domina cada vez más el coste temporal de la evaluación de la red neuronal. Por ejemplo, supongamos un vocabulario de $|\Omega| = 100$ palabras. Si tenemos una capa oculta con $|\bar{H}| = 10$,

el número de conexiones a evaluar en la capa de salida es $(|\Omega| + 1) \cdot |\bar{H}| = 1\,010$. Pero si tenemos un vocabulario de $|\Omega| = 10\,000$ palabras, un valor muy utilizado en la literatura al aplicar la técnica de la Short-List, y una capa oculta de $|\bar{H}| = 128$, el número de conexiones asciende a $1\,290\,000$. Este número es excesivo si además tiene que ejecutarse miles o cientos de miles de veces para guiar la búsqueda del sistema de reconocimiento/traducción donde se usan.

Al tratarse de modelos cuya entrada es simbólica (índices a una tabla que representa las palabras del vocabulario), el número de entradas diferentes y, por tanto, de constantes softmax que podemos llegar a precisar durante el proceso de decodificación es un número finito, aunque en la práctica excesivamente grande como para tenerlas todas. Observemos que en un NN LM de orden N hay $|\Omega|^{N-1}$ constantes diferentes, ya que a la entrada la red neuronal recibe $N - 1$ palabras del vocabulario Ω.

Nuestra propuesta consiste en precalcular las constantes de normalización softmax *más probables de aparecer durante el proceso de evaluación y almacenarlas en una tabla*. De esa manera, cuando hay que calcular una probabilidad tan sólo hay que ir y buscar la constante de normalización en la tabla precalculada. Sin embargo, existe la posibilidad de que la constante que necesitemos no haya sido precalculada. Se puede resolver este problema de dos formas diferentes:

- Calculando al vuelo la constante y utilizándola después, de manera que podemos guardar dicha constante para el futuro si lo creemos necesario. Denominaremos a esta aproximación on-the-fly Fast NN LM. Nótese que esta aproximación es igual a un NN LM estándard, pero acelerado mediante el precálculo de constantes de normalización softmax. En la experimentación ambas denominaciones se utilizarán de forma equivalente.

- Aplicando algún tipo de suavizado, como por ejemplo saltar a un modelo de N-gramas estándar, o a un NN LM de orden menor. A esta aproximación la denominaremos smoothed Fast NN LM.

4.1.2. Estado del arte

Este problema del coste del cálculo de las constantes de normalización de la función softmax ya había sido constatado con anterioridad en la literatura [Morin & Bengio, 2005; Schwenk, 2007; Bengio & Sénécal, 2008] y abordado de formas diferentes:

- Morin & Bengio [2005] proponen una descomposición jerárquica de la salida de la red neuronal, de manera que es posible calcular la probabilidad de una única palabra en $O(\log|\Omega|)$. Esta idea ha sido utilizada con buenos resultados en [Hai-Son *et al.*, 2011]. La ventaja de esta aproximación es que es posible utilizarla de forma combinada con las ideas que contaremos en las siguientes secciones, y además sirve tanto para acelerar el entrenamiento como la evaluación. También permite el uso de los modelos en sistemas totalmente acoplados.

- Schwenk [2007] propone dos formas de mejorar la eficiencia:

    1. Decrementar el tamaño de la salida eliminando las palabras menos frecuentes en la partición de entrenamiento. A esta lista de palabras se le conoce como Short-List y requiere el uso de un modelo de lenguaje adicional con el que realizar algún tipo de suavizado para calcular la probabilidad de las palabras eliminadas

(véase la sección 3.1.2). Esta aproximación obtiene una gran mejora en el coste computacional, sin embargo los tamaños útiles de la capa de salida siguen siendo demasiado grandes para permitir su integración acoplada en el sistema.

2. Agrupar juntos los N-gramas que comparten prefijos de tamaño $N-1$, de manera que con un único forward de la red se pueden calcular todos ellos. Esto también es factible cuando se utilizan funciones de activación diferentes a la softmax, ya que reduce en cualquier caso el número de forwards de la capa oculta de la red. La utilización de esta técnica en un sistema integrado no es totalmente eficiente, ya que es imposible conocer todos los N-gramas que van a generarse. No obstante los algoritmos de búsqueda se pueden implementar tratando de retrasar lo máximo posible la aplicación del modelo de lenguaje, y así poder juntar el mayor número posible de accesos al modelo de lenguaje.

- Bengio & Sénécal [2008] utilizan un estimador estocástico basado en técnicas de Montecarlo para estimar de forma aproximada el gradiente de la salida de la red neuronal. Esta idea es aplicable al entrenamiento de la red neuronal, logrando mejorar la eficiencia computacional, si bien la estabilidad del algoritmo de entrenamiento se reduce debido al error introducido por este estimador.

4.2 Precómputo de constantes de normalización softmax

El desarrollo presentado aquí se basa en un principio muy simple:

Teorema 1 *Un NN LM para bigramas tan sólo necesita $|\Omega|$ constantes de normalización softmax para cubrir todo el espacio de probabilidades del bigrama ($|\Omega|^2$).*

Nótese que todas las constantes de normalización necesarias para el bigrama se pueden calcular y guardar en una tabla de tamaño $|\Omega|$.

4.2.1. Entrenamiento de los modelos smoothed Fast NN LM

Se decide una jerarquía de modelos de lenguaje que comenzará en la N más grande, e irá descendiendo la N hasta llegar al bigrama. Por ejemplo, podríamos elegir 4-gramas, 3-gramas y por último el bigrama. Otra posibilidad es utilizar en el último lugar de esta jerarquía un modelo de N-gramas estándar.

Después, los NN LMs se entrenan según se ha explicado en el capítulo 2. Tras el entrenamiento se procede a precomputar las constantes de normalización softmax. Primero calculamos *todas* las constantes posibles para el bigrama. Después, para cada uno de los modelos con $N > 2$ calcularemos las constantes para los N-gramas más frecuentes en la partición de entrenamiento. También podría calcularse para *todos* los N-gramas del entrenamiento si su número no es excesivamente grande y, tanto el tiempo de cómputo como el espacio necesario en memoria para guardar las tablas lo permite.

Finalmente podemos hacer mixturas o cualquier otro tipo de combinación de modelos.

4.2.2. Detalles formales de la técnica de aceleración

El objetivo útil de un NN LM es calcular el logaritmo de la probabilidad de una secuencia de palabras, ya que durante el proceso de decodificación usaremos siempre el logaritmo de la probabilidad. Para ello, se definen las siguientes ecuaciones:

$$\log Z_I = \log \sum_{j=1}^{|\Omega|} exp(a_j) \quad (4.1)$$

$$\log o_i = \log \frac{exp(a_i)}{Z_I} = a_i - \log Z_I \quad (4.2)$$

$$\log p(\omega_i|\bar{h}_i) = \log o_{\omega_i}, \quad (4.3)$$

siendo $\log Z_I$ el logaritmo de la constante de normalización softmax asociada a la entrada $\bar{I}$ de la red neuronal correspondiente a la historia $\bar{h}_i$, y $\log o_i$ el logaritmo de la probabilidad para la palabra asociada a la neurona de salida i. La constante $\log Z_I$ puede estar almacenada en una tabla o ser calculada en el momento siguiendo la ecuación (4.1), o bien aplicar la aproximación basada en descender a un modelo más sencillo. Destacaremos que al trabajar con logaritmos, el cálculo de la ecuación (4.3) se reduce a calcular a_i con su correspondiente producto escalar del vector de entradas por el vector pesos correspondiente, más una resta por la constante Z_I, tal y como se muestra en la ecuación (4.2). Por otro lado, dada una entrada de la red neuronal y un conjunto de palabras de salida, es posible compartir cálculos computando la capa oculta una única vez para cada entrada diferente. Asumiremos que esto es posible hacerlo para acelerar el cálculo, pero lo obviaremos en el resto del desarrollo por simplicidad.

Las constantes de normalización softmax se guardan asociadas a nodos del TrieLM, guardando las secuencias de N-gramas de cada constante como transiciones persistentes en la estructura de datos. Una tabla asocia a cada estado destino de dicho N-grama en el TrieLM el valor de las constantes de normalización softmax.

4.2.3. Evaluación rápida de los modelos

Para evaluar los modelos se sigue este proceso:

1. Para cada posible valor de k, desde el N-grama de mayor orden hasta el bigrama:
2. Se busca el prefijo $k-1$ en la tabla de constantes de normalización softmax asociada al modelo M_k.
3. Si se encuentra la constante, se calcula la probabilidad buscada con el modelo M_k, y se devuelve.
4. En otro caso, se desciende k al modelo inmediatamente anterior, siguiendo un orden descendiente de N. Volver al paso 2 hasta que la probabilidad pueda ser calculada.

El algoritmo de evaluación ha sido implementado siguiendo la interfaz de modelos de lenguaje vista en la sección 3.2.1. El sistema de reconocimiento genera las hipótesis de manera que es posible agrupar aquellas que comparten una misma historia $\bar{h}_i$. La evaluación del NN LM se divide en dos partes:

1. Búsqueda de la constante de normalización softmax necesaria para calcular el logaritmo de la probabilidad de todo el vocabulario dada la historia $\bar{h}_i$, así como el modelo para el cual se conoce dicha constante (véase algoritmo 4.1 correspondiente al método **`prepareLM`**). En este punto se pueden dar estos cuatro casos:

   a) que el modelo para el que se quiere buscar la constante sea un N-grama estándar, en cuyo caso no se hace nada;

   b) que se haya elegido la opción de generar las constantes *al vuelo*, con lo que si la constante no se ha encontrado, será calculada;

   c) que no se encuentre la constante, ni se cumpla ninguna de las dos anteriores, esto precisará que se utilice un modelo más simple;

   d) que se encuentre la constante.

2. Cálculo del logaritmo de la probabilidad condicional de la palabra y_i dado su contexto. Este cálculo se reduce a la operación de la ecuación (4.2), calculando únicamente la activación de la neurona a_{y_i} que se corresponde con la palabra de entrada y_i, en lugar de calcular toda la capa de salida, gracias a que disponemos de la constante de normalización softmax. Esta parte se corresponde con el método **`getLMprob`**.

Por el teorema 1, se puede asegurar que el proceso siempre termina y encuentra la probabilidad asociada al N-grama de entrada, si utilizamos como modelo más simple un bigrama conexionista. De forma más general, podemos situar en la parte más baja de la jerarquía un modelo de N-gramas estándar, lo que de nuevo nos asegura poder calcular la probabilidad de cualquier palabra.

4.3 Integración eficiente de NN LMs en sistemas de reconocimiento/traducción

Además de la técnica que acabamos de exponer, para poder extraer el máximo rendimiento computacional a los NN LMs durante el proceso de decodificación, integraremos en el sistema dos tablas intermedias tipo caché. Estas tablas se acceden mediante el valor de un LMkey y guardan la información que precisamos. Son tipo caché porque cuando un LMkey produce una colisión con otro que ya estaba antes en la tabla, el valor anterior se machaca con el nuevo, perdiéndose. Estas tablas no crecen en tamaño, lo que permite que la memoria no se dispare.

Tenemos dos tablas intermedias:

1. Caché de probabilidad de N-gramas: guarda asociado al par $(\text{LMkey}, \omega \in \Omega)$ la probabilidad $p(\omega|\bar{h})$, siendo $\bar{h}$ la historia asociada al LMkey correspondiente. Sirve para evitar lanzar el forward de la red neuronal, y evitar el cálculo de la neurona de salida asociada a ω. Es una caché grande, con un tamaño de 512 mil entradas aproximadamente. Nótese que incluso para vocabularios pequeños, por ejemplo de $|\Omega| = 100$ palabras, el número de entradas necesarias en la caché para representar todo el conjunto posible de N-gramas asciende a $|\Omega|^N$, que para $N = 3$ son $100^3 = 1\,000\,000$ de entradas.

Algoritmo 4.1 Busca la constante de normalización y el modelo M_k, dada una secuencia de $N-1$-grama palabras. *// It searchs the normalization constant and its corresponding model M_k, given a sequence of $N-1$-gram words. Referenced at page 302.*

Require: An input sequence of $N-1$ words that are the prefix of the N-gram ending at input word y_i, a hierarchy of m language models

Ensure: Search or compute the softmax normalization constant needed to compute the conditional probabilities and return the corresponding model M_k and their softmax normalization constant lZ_I if needed

1: **function** prepareLM for NN LMs $(y_{i-N+1}, y_{i-N+2}, \ldots, y_{i-1})$
2: **begin**
3: Let $k = m$, the last hierarchy model index
4: **while true do**
5: **if** M_k is a standard N-gram **then**
6: **return** M_k
7: **end if**
8: Let $n' = N$-gram order of model M_k
9: Let $lZ_I =$ logarithm softmax normalization constant of $y_{i-n'+1}y_{i-n'+2}\ldots y_{i-1}$ for model M_k
10: **if** compute on-the-fly $\wedge$ $lZ_I =$ **nil then**
11: Do forward pass of the NN LM M_k
12: Update $lZ_I =$ compute softmax normalization constant for M_k
13: **end if**
14: **if** $lZ_I \neq$ **nil then**
15: Do forward pass of the NN LM M_k, except the output layer
16: **return** M_k, lZ_I
17: **else**
18: $k =$ previous model index in hierarchy
19: **end if**
20: **end while**
21: Report output error: "softmax normalization constant not found"
22: **end function**

Tabla 4.1: *Características del corpus LOB-ale utilizado para validar la perplejidad de los modelos smoothed Fast NN LM. //* LOB-ale corpus chracteristics.

Set	# sentences	# running words
Training	4 303	37 606
Validation	600	5 348
Test	600	5 455

2. Caché de unidades ocultas: guarda asociado a un LMkey la activación de las unidades ocultas de la red neuronal. La capa oculta suele tener tamaños que oscilan entre las 200 y las 500 neuronas, por lo que el tamaño de esta caché será más reducido que la anterior. Concretamente, tiene aproximadamente 32 mil entradas.

Todo esto se une a la técnica de precalcular constantes de normalización softmax, permitiendo acelerar lo suficiente la evaluación de NN LMs como para poder integrarlos en el proceso de decodificación.

4.4 Experimentación

4.4.1. Corpus LOB-ale

El corpus para esta tarea es una partición aleatoria de frases extraídas del LOB corpus [Johansson *et al.*, 1978]. El tamaño de la partición es lo suficientemente pequeña para contener un vocabulario reducido que permita entrenar NN LMs sin hacer uso de la Short-List. Además, el vocabulario será cerrado, con lo que obviaremos el problema de las palabras desconocidas cuando se intentan comparar modelos de lenguaje diferentes. En la tabla 4.1 se pueden ver las características de las particiones elegidas.

4.4.2. Comparando NN LMs con SRI

Se han entrenado tres NN LMs de diferente tamaño en la capa de proyección y en la capa oculta: 80-128, 128-192, 192-192. Se han estimado bigramas, 3-gramas y 4-gramas para cada tamaño. Los parámetros del algoritmo de entrenamiento han sido:

- learning rate de 0.002,
- momentum de 0.001,
- y weight decay de 1×10^{-9}.

Una vez entrenados los NN LMs, se ha calculado la PPL para el conjunto de validación, que se muestra en la tabla 4.2. También se muestran los resultados de PPL con el modelo Mixed NN LM, que es una combinación lineal de los anteriores.

También se han entrenado modelos de N-gramas estándar usando el SRI toolkit [Stolcke, 2002]. Los N-gramas estándar se han estimado sobre la concatenación de los conjuntos de entrenamiento y de validación. Se pueden ver y comparar los resultados de PPL sobre el test en la tabla 4.3. Se puede observar que el modelo Mixed NN LM obtiene una mejora de 3.29 puntos absolutos, una mejora del $4\,\%$ en la perplejidad de los modelos. La diferencia es más

Tabla 4.2: *Perplejidad con el conjunto de validación para los NN LMs estimados. //* Perplexity of the validation set.

Language Model	2-gram	3-gram	4-gram
NN LM 80–128	82.17	73.62	74.07
NN LM 128–192	82.52	73.30	72.50
NN LM 192–192	80.01	71.90	71.91
Mixed NN LM	78.92	71.34	70.63

Tabla 4.3: *Perplejidad con el conjunto de test para los mejores NN LMs y los N-gramas estándar. //* Perplexity of the test set.

Language Model	2-gram	3-gram	4-gram
SRI	88.29	**83.19**	84.41
NN LM 192–192	89.77	82.11	**81.57**
Mixed NN LM	88.72	80.94	**79.90**
Mixed NN LM + SRI-3-grama	–	–	**68.52**

grande conforme aumentamos el tamaño de la N, mostrando la ventaja clara de los NN LMs para estimar modelos de mayor orden con corpus de tamaño reducido.

Es interesante observar que la PPL de los NN LMs mejora al modelo estadístico estándar. Este comportamiento ha sido observado en experimentos previos a esta tesis cuando el vocabulario de la tarea tarea era cerrado.

Para cerrar este primer apartado de experimentación, se ha combinado el Mixed NN LM con el 3-grama SRI estándar, obteniendo una PPL en el test de 68.52, lo que supone:

- una mejora de 14.67 puntos absolutos de PPL, $17\,\%$ de mejora sobre los 83.19 puntos de PPL del 3-grama estándar de SRI;
- una mejora de 11.38 puntos absolutos de PPL, $14\,\%$ de mejora sobre los 79.90 puntos de PPL del Mixed NN LM.

Estos resultados se han obtenido siguiendo la aproximación de NN LMs estándar, sin aplicar todavía la técnica de aceleración explicada en este capítulo. Los usaremos como punto de referencia para el resto de experimentos.

Conclusión

Se ha probado la configuración estándar de NN LMs en una tarea de tamaño reducido, lo que permite compararlos con N-gramas estadísticos estándar de forma equitativa utilizando ambos modelos el mismo vocabulario. El resultado ha sido que los NN LMs mejoran a los N-gramas estadísticos estándar en el escenario de esta experimentación. Además, la combinación de NN LMs con el 3-grama estándar de SRI obtiene una mejora del $17\,\%$ sobre el mejor resultado con N-gramas estándar, y del $14\,\%$ sobre el mejor de los NN LMs. Esta mejora es lo suficientemente amplia como para reflejarse en el resultado final de un sistema de reconocimiento/traducción donde estos modelos fueran utilizados.

Tabla 4.4: *Evaluación rápida del 4-grama Mixed NN LM. //* Fast evaluation of 4-gram Mixed NN LM.

Setup	ms/word	speed-up	test PPL
Mixed NN LM	6.43	0	79.90
On-the-fly Fast NN LM	1.82	3	79.90
Smoothed Fast NN LM	0.19	33	80.78
Smoothed-SRI Fast NN LM	0.19	33	79.02

4.4.3. Técnica de evaluación rápida con precálculo de las constantes

Además de las configuraciones expuestas anteriormente, se ha implementado y experimentado utilizando un modelo de N-gramas estadístico en la parte más baja de la jerarquía de modelos. Por tanto, se han probado estas otras configuraciones:

- **On-the-fly** Fast NN LM: calcula *al vuelo* las constantes de normalización softmax cuando estas no se encuentran;
- **Smoothed** Fast NN LM: siempre utiliza un NN LM más simple cuando no se encuentra una constante de normalización, así hasta llegar a los bigramas NN LMs;
- **Smoothed-SRI**:Fast NN LM: utiliza un modelo más simple cuando no se encuentra una constante de normalización, pero sustituyendo el último de los modelos, el bigrama NN LM, por un bigrama estadístico estándar estimado con SRI.

Los resultados obtenidos para cada aproximación de evaluación rápida, para el modelo 4-grama Mixed NN LMs entrenado para la tarea LOB-ale, se resumen en la tabla 4.4.

Configuración On-the-fly. Los resultados de este configuración no afectan a la PPL de los modelos, pues finalmente se calculan todas las constantes de normalización necesarias. La figura 4.1 muestra la evolución del coste temporal por palabra, así como el porcentaje de constantes precalculadas utilizadas, y el porcentaje de constantes que han tenido que calcularse *al vuelo*. Se observa que al utilizar el máximo de constantes de normalización (extraídas del conjunto de entrenamiento) es posible bajar desde 2.3 milisegundos/palabra a casi 1.8 en el caso de 4-gramas. Esta técnica alcanza un "speed-up" de 3 si la comparamos con el NN LM estándard (Mixed NN LM) Al usar 3-gramas, el coste obviamente se reduce, necesitando desde 2.1 milisegundos/palabra hasta 1.0 aproximadamente. Alcanza un speed-up de 2 si comparamos con el NN LM estándard. Sin embargo, *este speed-up no es suficiente* para sistemas donde se pretende integrar NN LMs durante la decodificación, más todavía cuando el valor de N crece por encima de 4-gramas.

Configuración Smoothed. Esta configuración sí afecta a la perplejidad de los modelos, como se puede apreciar en la tabla 4.4 y la figura 4.2. A mayor número de constantes precalculadas, más se acerca el valor de PPL del modelo al valor del 4-grama NN LM estándar. De este experimento cabe destacar que la pérdida de PPL observada es de 1 punto, mientras que en milisegundos/palabra se obtiene un speed-up de 33.

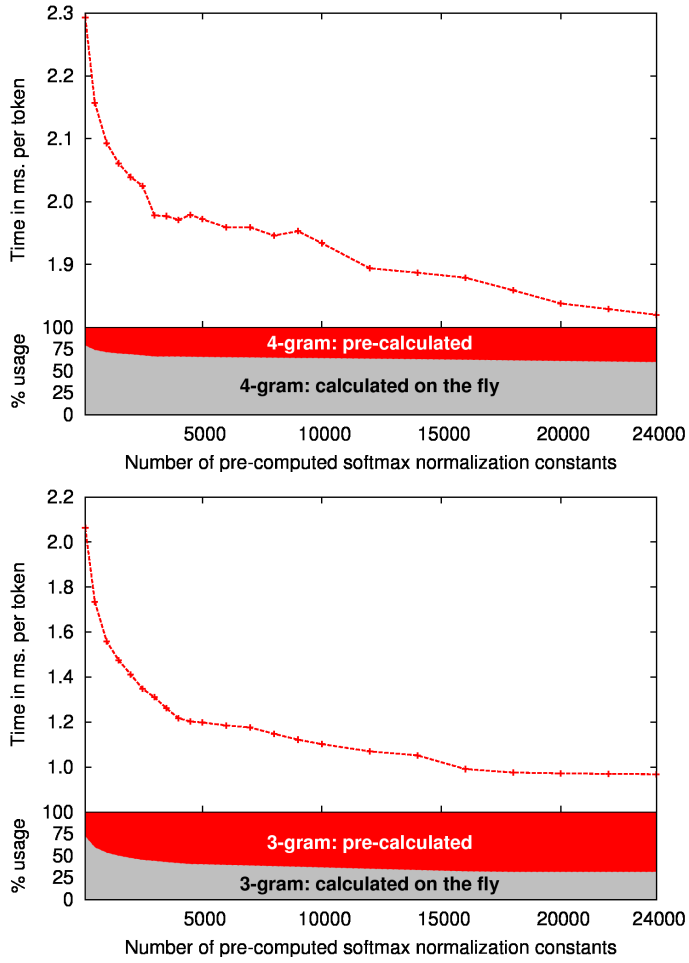


Figura 4.1: *Influencia del número de constantes de normalización softmax precalculadas en el tiempo de evaluación de los modelos para la configuración "On-the-fly". 4-grama Mixed NN LM (arriba) y 3-grama Mixed NN LM (abajo).* // Evaluation time versus number of softmax normalization constants for the "On-the-fly" configuration.

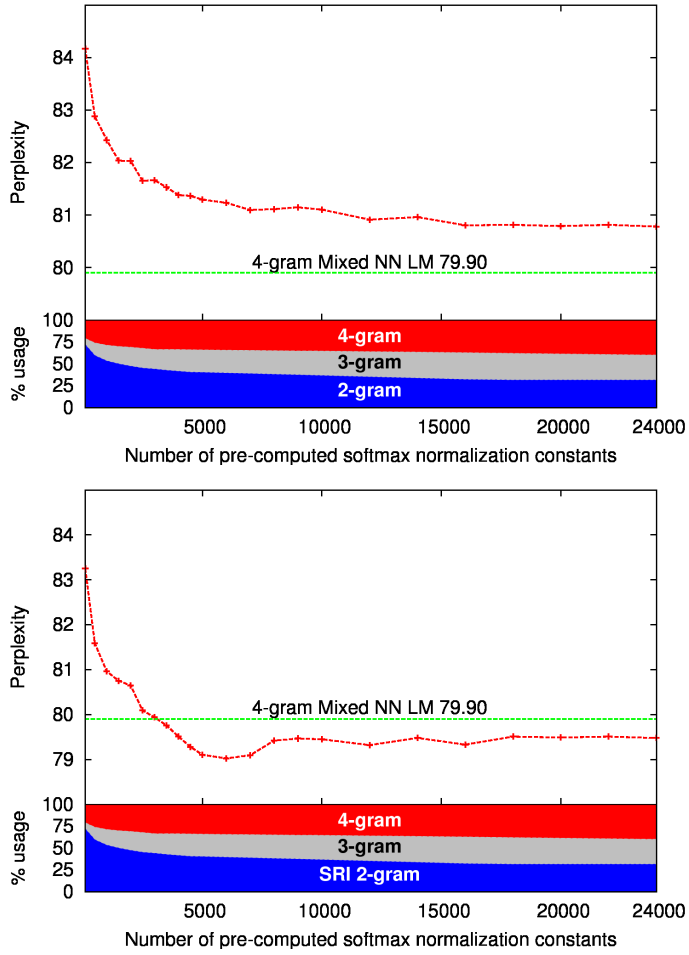


Figura 4.2: *Influencia en la perplejidad del test del número de constantes de normalización softmax precalculadas para las configuraciones Smoothed (arriba) y Smoothed-SRI (abajo). //* Test set PPL versus number of softmax normalization constants.

Configuración Smoothed-SRI. Sorprendentemente esta configuración mejora la PPL del sistema. La hipótesis más factible es que los N-gramas que contienen palabras fuera del vocabulario del Training no han sido correctamente entrenados, y entonces la proyección de dichas palabras no es buena en la entrada de los NN LMs. Dada esta situación, los heurísticos de los modelos de lenguaje estadísticos estándar, para el caso de bigramas, funcionan un poco mejor y permiten avanzar en 0.9 puntos de PPL el sistema, comparado con el mejor de los Mixed NN LM. La velocidad se mantiene igual al caso anterior, con lo que sigue habiendo un speed-up de 33.

Conclusión

Se han probado tres configuraciones distintas de Fast NN LMs, viendo el efecto del número de constantes de normalización en el resultado final, tanto a nivel de PPL como a nivel de velocidad. Podemos concluir que se ha logrado que el modelo vaya 33 veces más rápido que el estándar, con una pérdida de tan sólo 0.9 puntos de PPL o una ganancia de 0.9 puntos de PPL si se usa el bigrama estándar en la parte más baja de la jerarquía de modelos.

4.5 Comparando NN LMs integrados

Resumimos aquí resultados de experimentos realizados usando el sistema de reconocimiento de escritura que se detalla en el capítulo 8. Se han utilizado los modelos de lenguaje denotados por *4-grama* Θ *=21*. Véase la sección 8.7.1 para más detalles sobre los modelos.

4.5.1. Efecto de la integración de smoothed Fast NN LMs

Las figuras 4.3, 4.4 y 4.5 muestran una comparativa del efecto de la poda en el WER y el coste computacional para un experimento de escritura manuscrita (véase sección 8.7). En este experimento se compara el mejor de los modelos de N-gramas estándar con el mejor de los NN LMs entrenados para dicha tarea, integrando el NN LM a través de la idea del precálculo de las constantes de normalización softmax, creando un smoothed Fast NN LM que combina bigramas, 3-gramas y 4-gramas para acelerar la evaluación en el sistema integrado; también se ha comparado con un Fast NN LM que calcula la constante de normalización *al vuelo*, en caso de no encontrarla en la tabla. Podemos analizar estos resultados desde tres perspectivas.

Relación entre el WER y el tiempo. La figura 4.3 muestra un punto a partir del cual el sistema con smoothed Fast NN LMs presenta mejores resultados, en coste computacional, que el sistema con bigramas estándar para un mismo WER. Antes de llegar a ese punto, ambos sistemas tienen un coste muy parecido. En todo caso, a igualdad de poda (véase figura 4.5), el WER es mejor siempre para el sistema que integra smoothed Fast NN LMs. Se observa también que el NN LM estándar presenta un coste aproximadamente 7 veces mayor.[1]

Relación entre el WER y el tamaño de la poda estática. En la figura 4.4 podemos comparar la mejora en WER introducida por el smoothed Fast NN LM para cada valor de histograma de poda. La ventaja aumenta rápidamente conforme aumentamos el número de estados

[1] Hay que considerar que el NN LM estándar *también* utiliza la tabla de constantes de normalización softmax; en caso de no encontrar la constante, la calcula *al vuelo*.

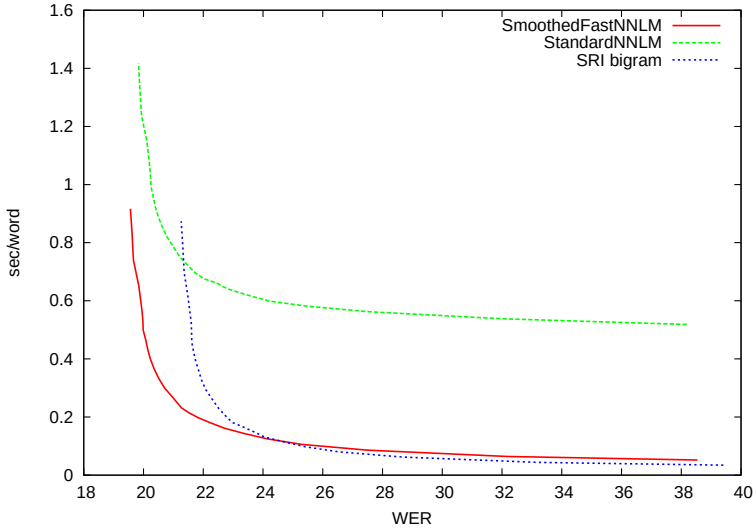


Figura 4.3: *Influencia del WER y el coste computacional de los sistemas integrados con smoothed Fast NN LM, NN LM estándar y N-gramas estándar en la tarea IAM-DB de reconocimiento de escritura manuscrita.* // Computational cost versus WER on the IAM-DB task. Referenced at pages 302 and 303 .

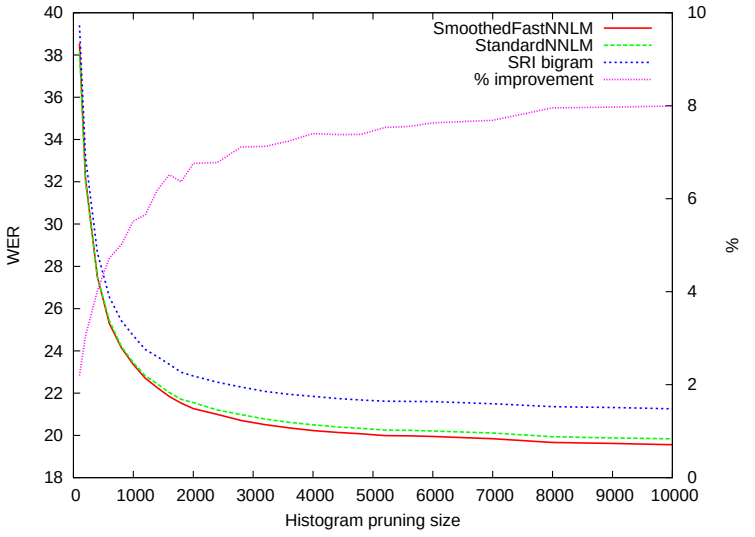


Figura 4.4: *Influencia de la poda en el WER en los sistemas integrados con smoothed Fast NN LM, NN LM estándar y N-gramas estándar en la tarea IAM-DB de reconocimiento de escritura manuscrita.* // WER versus histogram pruning values on the IAM-DB task. Referenced at pages 302 and 303 .

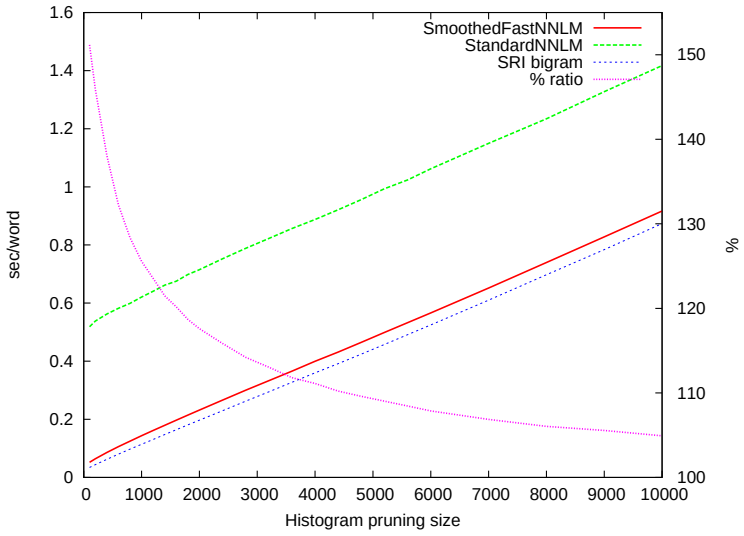


Figura 4.5: *Influencia de la poda en el coste computacional en los sistemas integrados con smoothed Fast NN LM, NN LM estándar y N-gramas estándar en la tarea IAM-DB de reconocimiento de escritura manuscrita.* // Computational cost versus histogram pruning values on the IAM-DB task. Referenced at pages 302 and 303 .

activos, llegando hasta casi un $8\,\%$ de mejora en WER lo que supone una diferencia absoluta de 1.7 puntos. Las diferencias entre el NN LM estándar y el smoothed Fast NN LM no son significativas, siendo favorable para el smoothed Fast NN LM por $\approx 1.5\,\%$ de WER (lo que supone ≈ 0.3 puntos absolutos). Además, el smoothed Fast NN LM permite mejorar el baseline de N-gramas estándar en un $8\,\%$ con un coste temporal comparable al de los N-gramas estadísticos estándar.

Coste computacional de los smoothed Fast NN LMs y NN LMs estándar. En la figura 4.5 podemos comparar el coste computacional del proceso de decodificación para distintos valores de poda estática. Como se puede ver, tanto el sistema estándar, como el que usa smoothed Fast NN LMs (y también el NN LM estándar), tienen un comportamiento lineal, cuantos más estados dejamos pasar, más coste tiene el proceso. La pendiente es muy parecida en todos los casos, y el coste de añadir smoothed Fast NN LMs a la búsqueda utilizando las ideas presentadas en este capítulo precisa un $\approx 50\,\%$ más de tiempo para la poda más fuerte (en torno a 100 estados activos). Este coste se va reduciendo hasta un $\approx 5\,\%$ más de tiempo para el valor con mayor número de estados activos vivos (10 000). Esto quiere decir que conforme aumentamos el número de estados activos, la diferencia en coste computacional entre ambos sistemas se reduce. Al aumentar el número de estados activos es de esperar que muchos de ellos supongan búsquedas en el modelo de lenguaje que comparten la historia del N-grama, y por tanto el coste del forward de las redes neuronales se amortiza entre todos ellos. La relación entre ambos costes sigue una curva parabólica, ya que existe una relación proporcionalmente inversa. Para valores de poda que ofrecen una relación entre WER y tiempo adecuada (en torno a los 4 000 estados activos) el tiempo para el sistema con smoothed

Fast NN LMs es un $\approx 10\,\%$ superior, mientras que el WER es un $\approx 7.5\,\%$ más bajo. No obstante, si nos fijamos en la recta definida por el NN LM estándar, observamos que el sistema es aproximadamente dos veces más lento, lo que supone un incremento del coste del $\approx 100\,\%$.

4.5.2. Conclusiones

La integración de smoothed Fast NN LMs, usando las técnicas de evaluación rápida aquí expuestas, precisa de un coste computacional algo mayor que el sistema que usa N-gramas estándar. Este incremento del coste se ve compensado por la ganancia en WER. Observamos claramente que la pequeña pérdida de calidad que tiene el sistema smoothed Fast NN LM frente al NN LM estándar compensa, ya que consigue que el sistema sea al menos dos veces más rápido. El sistema resultante puede ser configurado para conseguir uno de estos dos objetivos:

- Mejorar el coste computacional del sistema, ya que la integración permite obtener el mismo WER a un coste menor (véase la figura 4.3).
- Mejorar la calidad del sistema en un $8\,\%$, a cambio de un coste computacional entre un $5\,\%$ y un $10\,\%$ más alto (véanse las figuras 4.4 y 4.5).

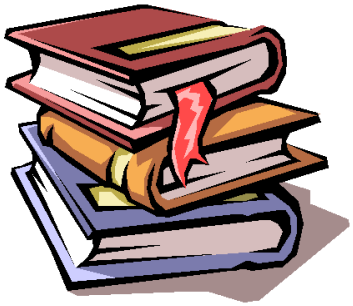
4.6 Resumen

Se ha presentando con detalle el algoritmo utilizado para acelerar la evaluación de los NN LMs y se ha probado empíricamente. Los objetivos logrados han sido:

- Presentación y formalización de la técnica que permite acelerar el cálculo de la función de activación softmax.
- Comparación de la técnica de aceleración desarrollada con la aproximación estándar, tanto a nivel de calidad de los modelos como a nivel de coste computacional. Esta primera experimentación dio lugar a la publicación [Zamora-Martínez *et al.*, 2009].
- Integración de los NN LMs en el proceso de decodificación y su evaluación en el marco del reconocimiento de escritura manuscrita. Esto ha dado lugar a resultados a nivel de WER muy interesantes, llegando a mejorar 1.7 puntos absolutos de WER, obteniendo un coste computacional casi idéntico a los modelos estándar de N-gramas. Se está preparando un artículo para la revista "IEEE Transactions on Neural Networks" donde se espera publicar estos resultados [Zamora-Martínez *et al.*, 2012b].

5

NN LMS CON CACHÉ



Índice

En este capítulo se detalla la extensión de los NN LMs para su adaptación al contexto a través de las ideas de los modelos de lenguaje con caché [Kuhn & Mori, 1990]. A este nuevo modelo lo llamaremos Cache NN LM. Los objetivos de este capítulo son:

- Resumir el estado del arte en modelos de lenguaje con caché.
- Formalizar el modelo Cache NN LM y discutir cómo puede adaptarse la caché para su utilización en tareas de reconocimiento de secuencias o de traducción.

5.1 Motivación

El objetivo de un modelo de lenguaje es aportar información lingüística sobre la construcción de una oración que se combina dentro del sistema de reconocimiento. Esta información típicamente proviene de corpus textuales de tamaño considerablemente grande, donde es posible obtener modelos fehacientes de una determinada tarea o dominio, e incluso modelos sin restricción del dominio. Sin embargo, durante un discurso hablado o documento escrito que se quiera transcribir, o bien traducir, el modelo de lenguaje carece de conocimiento sobre el contexto cercano a la frase que se esté procesando en este momento. Es muy habitual que, en un documento, palabras que han aparecido en las frases anteriores vuelvan a aparecer, ya que si el documento versa sobre un determinado tema se van a repetir palabras para referirse a las entidades nombradas anteriormente. En otro casos, la aparición de una determinada palabra puede disparar la probabilidad de otras diferentes.

5.2 Modelos de lenguaje con caché

Los modelos de lenguaje de N-gramas estiman la probabilidad de una palabra conocidas únicamente las $N-1$ anteriores. Tomando esta problemática en consideración surgen los modelos con caché [Kuhn & Mori, 1990], con la intención de introducir información sobre el contexto actual del discurso que está procesando la máquina.

5.2.1. Antecedentes

En este primer trabajo [Kuhn & Mori, 1990], el modelo caché se estima como la combinación de dos modelos estadísticos:

- el modelo de lenguaje general $p(y_j|y_{j-1}y_{j-2}y_{j-N+1})$,
- el modelo de lenguaje específico para la caché $p_{cache}(y_j|\bar{h}_1^{j-1})$,

siguiendo la ecuación:

$$p(y_j|y_{j-1}\dots y_1) = \alpha p(y_j|y_{j-1}y_{j-2}y_{j-N+1}) + (1-\alpha)p_{cache}(y_j|\bar{h}_1^{j-1})\,, \qquad (5.1)$$

donde $\bar{h}_1^{j-1}$ es el resumen que sirve como caché para el modelo específico.

En esta línea se desarrollan en [Rosenfeld, 1996] los modelos con disparador (triggers), donde la memoria caché del modelo relaciona palabras, de forma que la aparición de una palabra hace que se dispare la probabilidad de aparición de otra u otras totalmente diferentes.

Otros autores [Florian & Yarowsky, 1999; Chien & Chueh, 2011] proponen calcular las probabilidades del modelo de lenguaje incluyendo como representación de la historia de las palabras distribuciones de probabilidad dependientes del dominio. Cada dominio d contribuye de forma diferente a la probabilidad del modelo de N-gramas en función del contexto:

$$p(y_j|y_{j-1}\dots y_1) = \sum_k p(d_k|y_{j-1}\dots y_1)p_d(y_j|y_{j-1}\dots y_{j-N+1})\,. \qquad (5.2)$$

Todas estas aproximaciones permiten modificar las probabilidades del modelo de lenguaje general usando modelos contextuales especializados, donde parte de la información de estos

modelos contextuales se extrae del propio documento a analizar. En muchas ocasiones la literatura se refiere a este modelado como *adaptación* del modelo de lenguaje.

5.2.2. Reducción del problema de la dispersión de los datos

La dispersión de los datos es un problema crucial en la estimación de los parámetros de los modelos usados para la adaptación. Una posible solución es formar clusters de la historia de las palabras y usar estos clusters para estimar los modelos. También es posible abordar el problema de la dispersión de los datos a través del "Latent Semantic Analysis" [Landauer & Dumais, 1997]. La descomposición en valores singulares permite reducir la dispersión [Berry *et al.*, 1995]. Estos modelos asumen que palabras semánticamente parecidas influyen de forma similar en la predicción de nuevas palabras [Coccaro & Jurafsky, 1998; Bellegarda, 2000].

En [Hofmann, 1998; Blei & Jordan, 2002] introducen el "probabilistic Latent Semantic Analysis" creando conjuntos de tópicos ocultos contenidos en un modelo generativo. En este modelo, los documentos se representan como distribuciones sobre los tópicos. La distribución de probabilidad de las palabras se estima para cada tópico.

En [Mitchell & Lapata, 2009] utilizan composiciones de modelos de palabras basados en vectores para representar el contexto. Asumen que los vectores utilizados representan el contenido semántico de los datos.

En [Afify *et al.*, 2007] proponen otra forma para reducir el efecto de la dispersión de los datos. Basándose en que es posible adaptar un modelo acústico complejo a través de unas pocas iteraciones con datos del domino, proponen representar las palabras en un espacio continuo y aprender en dicho espacio una mixtura de Gaussianas. De esa forma, es posible aplicar un algoritmo tipo "Maximum Likelihood Linear Regression" [Legetter & Woodland, 1995] para adaptar el modelo de lenguaje al nuevo dominio.

En [Chien & Chueh, 2011] presentan un modelo de lenguaje de "Dirichlet" basado en clases. Cada clase se asocia con una palabra del contexto. Las cuentas de las clases en el contexto sirven para estimar de forma dinámica una combinación entre probabilidades del del dominio y probabilidades de las palabras dentro de dicho dominio.

Otra forma de evitar el problema de la dispersión de los datos es utilizar modelos que permitan una interpolación de los eventos no vistos en entrenamiento, como por ejemplo las redes neuronales. En esa dirección van los NN LMs ideados y descritos en esta tesis. No obstante, es necesario modificar el modelo conexionista original para trabajar con información de contexto a larga distancia.

5.3 Extensión de NN LMs con caché

Una de las ventajas de los NN LMs es su flexibilidad para añadir información a la entrada o la salida de la red neuronal. Denominaremos Cache NN LM a un modelo conexionista de lenguaje que de alguna forma introduce información del contexto del discurso al que pertenece la frase que se está decodificando. Denominaremos interacción del sistema con el usuario a una secuencia de frases del usuario y su correspondiente transcripción y/o respuesta del sistema. Las frases dentro de una interacción se asume que están relacionadas ya que pertenecen al mismo discurso, mientras que las frases entre una interacción y otra no tienen por qué. Por esta razón, la caché se inicializará cada vez que empiece una nueva interacción.

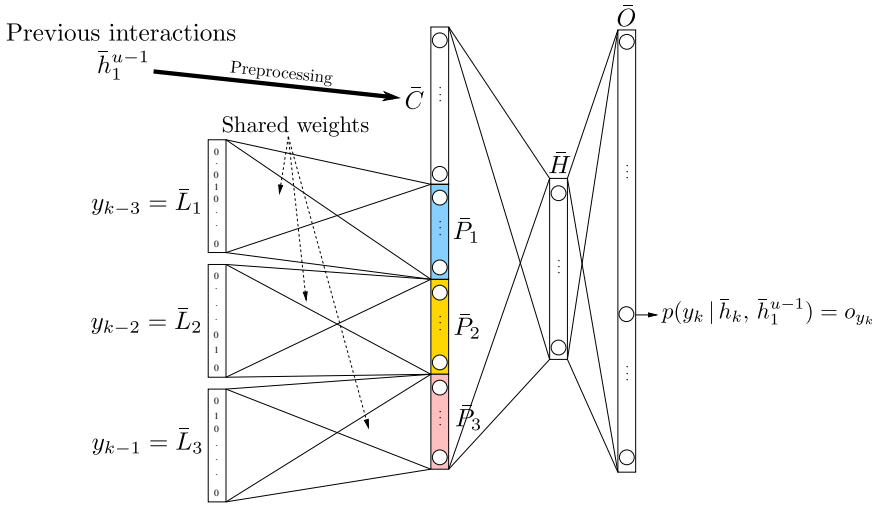


Figura 5.1: *Cache NN LM. La entrada del NN LM se extiende con el vector $\bar{C}$ que incluye información sobre el contexto representando en forma de* bolsa de palabras. // Cache NN LM. $\bar{C}$ includes context information as a "bag-of-word" vector. Referenced at page 304.

Si denotamos con u el índice de la frase actual dentro de la interacción actual, el modelo Cache NN LM extiende la ecuación (2.2) añadiendo un parámetro $\bar{h}_{j,1}^{u-1}$ que da cuenta de la caché. Este parámetro es un resumen formado por las últimas palabras desde la primera frase de la interacción actual hasta la penúltima. Por simplicidad denotaremos $\bar{h}_{j,1}^{u-1}$ como $\bar{h}_1^{u-1}$, ya que es independiente del índice j de la palabra en la frase actual:

$$p(\bar{y}) \approx p(\bar{y}|\bar{h}_1^{u-1}) \approx \prod_{j=1}^{|\bar{y}|} p(y_j|\bar{h}_j, \bar{h}_1^{u-1}) \,. \tag{5.3}$$

El parámetro $\bar{h}_1^{u-1}$ se codifica en la entrada de la red neuronal como un vector de características $\bar{C}$, tipo *bolsa de palabras*, tal y como se ilustra en la figura 5.1. Dicho vector se actualiza con información sobre el contexto en el cual se encuentra la frase actual.

5.4 Codificación del contexto en la caché del NN LM

Podemos distinguir en este caso tres tipos de fuentes de información que pueden componer la caché $\bar{h}_1^{u-1}$:

- **Información conocida:** son eventos que suceden en el proceso y que son conocidos. Por ejemplo, en un sistema de traducción automática las frases correspondientes a la lengua origen del documento que queremos traducir son este tipo de información. Salvo errores en los documentos originales, esta información se asume como correcta.

- **Información con, posiblemente, errores:** son las propias frases reconocidas por el sistema. Pueden contener errores de transcripción.

- **Información dependiente:** este tipo es aplicable, por ejemplo, en sistemas de diálogo, y se refiere a las frases que el sistema de diálogo da como respuesta al usuario. Estas frases no contienen errores a nivel lingüístico, pero son respuesta a frases del usuario que pueden haberse reconocido de forma incorrecta, y por tanto son respuestas del sistema que no son necesariamente correctas desde el punto de vista de la comprensión completa del diálogo con el usuario.

El vector $\bar{C}$ está compuesto por un conjunto de neuronas de entrada que se conectan directamente con la capa oculta $\bar{H}$, mediante la matriz de pesos $\mathbf{W}_{C,H}$ (véase figura 5.1). Este conjunto de neuronas se asocia con las unidades lingüísticas de nuestra tarea, que pueden ser tanto palabras del modelo de lenguaje, como palabras de la lengua origen (en el caso de traducción), o respuestas del sistema (en el caso de sistemas de diálogo), entre otras. Cada unidad lingüística se asocia con una o más neuronas del conjunto $\bar{C}$. El procedimiento de cálculo de la activación de las neuronas de entrada $\bar{C}$ es:

1. Cada vez que se introduce una unidad lingüística en el conjunto, las neuronas implicadas se activan con valor 1, y la activación del resto de neuronas se multiplica por una constante $\alpha < 1$.

2. Todas las activaciones que tras aplicar estas dos operaciones sean menores que un determinado umbral ϵ, se fijan a 0. El valor de α y de ϵ permite controlar la longitud de la historia de la interacción con el usuario que queremos representar. Para valores de $\alpha = 0.95$ y $\epsilon = 0.00001$ la longitud de la historia es de aproximadamente 270 palabras.

3. La caché se actualiza tras finalizar la transcripción de cada frase, y se procesan todas las unidades lingüísticas nuevas en orden de izquierda a derecha, tomando el estado actual de $\bar{C}$ como punto de partida.

4. Cuando termina una interacción con el usuario (que está compuesta por una secuencia de frases/respuestas), la caché se inicializa a cero.

Veamos un ejemplo sencillo de actualización del conjunto de neuronas $\bar{C}$. Supongamos que tenemos un sistema de reconocimiento automático del habla para un entorno de transcripción con el vocabulario $\Omega = \{a, b, c\}$. La caché estaría formada por 3 neuronas, de manera que tenemos una neurona a, otra b y otra c, que representan a su respectiva palabra del vocabulario Ω. Con un valor de $\alpha = 0.95$ y la siguiente secuencia de frases reconocidas por el sistema, la caché tomaría los siguientes valores:

	$\bar{h}_1^{u-1} = \bar{C}$			$\bar{y}^{(u)}$
u	a	b	c	Frase transcrita
0	0	0	0	$a\ b\ a\ b$
1	0.95	1	0	$c\ a$
2	1	0.95^2	0.95	$c\ b\ a\ c$
3	0.95	0.95^2	1	...
⋮	⋮	⋮	⋮	⋮

En la tabla se observa como la caché se actualiza tras cada transcripción. El contenido de cada fila es el estado del conjunto $\bar{C}$ durante la transcripción completa de la frase actual.

Algoritmo 5.1 Calcula al vuelo durante el entrenamiento la entrada $\bar{C}$ de un Cache NN LM. *// On-the-fly computation of input $\bar{C}$ for Cache NN LM during training. Referenced at page 304.*

Require: The set of user/machine sentences previous to current sentence u, maximum history distance X at the cache

Ensure: Compute on-the-fly the cache input vector during neural network training

1: **function** computeCacheInput($\bar{y}^{(1)}, \bar{y}^{(2)}, \ldots, \bar{y}^{(u-1)}, X \in \mathbb{N}$)
2: **begin**
3: Let C an array of size $|\bar{C}|$ initialized with zeroes, and let $c(i)$ its i-th component
4: Let num $= 0$ an auxiliar counter variable
5: **for all** words $\omega \in \{\bar{y}^{(j-1)}, \ldots, y^{(2)}, y^{(1)}\}$ in descending order, while num$< X$ **do**
6: **for all** input cache neuron $p \in$ neurons associated with word ω **do**
7: $c(p) = \max\{c(p), pow(\alpha,\ \text{num})\}$
8: num $=$ num $+1$
9: **end for**
10: **end for**
11: **return** $\bar{C}$
12: **end function**

Los conjuntos de datos utilizados durante el entrenamiento se extienden con un token especial que sirve para indicar que una interacción ha terminado. Llegado ese caso se vacía la caché. Al entrenar, el valor de $\bar{C}$ es calculado *al vuelo* con cada patrón, con un coste lineal con el número de neuronas en $\bar{C}$ y lineal con la longitud máxima de la historia (asumiendo que se asocia cada palabra del vocabulario a una única neurona). La longitud de la historia depende de los parámetros α y ϵ, de tal forma que podemos definirla como $X = \dfrac{\log \epsilon}{\log \alpha}$. El algoritmo 5.1 especifica este procedimiento.

Finalmente el coste de un paso del BP añadiendo caché al NN LM es $O(|\bar{C}| + X + |\bar{W}| + |\bar{B}|)$, siendo $\bar{W}$ el conjunto de *todos* los pesos de la red, y $\bar{B}$ el conjunto de todos los bias. El coste del algoritmo del BP estándar es $O(|\bar{W}| + |\bar{B}|)$. Tanto los Cache NN LMs como los NN LMs pueden llegar a tener millones de conexiones, mientras que $X \approx 270$ y $|\bar{C}| \approx 10\,000$, si bien este último depende de la tarea, el vocabulario, y la asociación de palabras del vocabulario a neuronas en la caché. Debido a esta diferencia en magnitud, teóricamente podemos despreciar el coste computacional debido a la caché.

5.5 Resumen

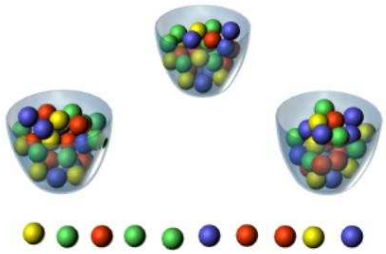
Se han extendido los NN LMs para permitir la introducción de un módulo con memoria caché que permite adaptar las probabilidades del modelo en función de la historia a largo plazo. Este modelo ha sido aplicado a tareas de comprensión ("Speech Language Understanding", SLU) en un sistema de diálogo, y ha sido publicado en el ICASSP del 2012 [Zamora-Martínez *et al.*, 2012a]. El capítulo 7 describe todos los detalles referentes a esta experimentación.

Parte II

Aplicación de NN LMs al reconocimiento de secuencias

6

FORMALIZACIÓN DE LOS HMMS Y DECODIFICACIÓN



Índice

Este capítulo presenta la formalización de los modelos matemáticos utilizados en los sistemas de reconocimiento de habla y escritura, así como la descripción de los algoritmos utilizados para llevar a cabo la decodificación de una frase.

Los modelos ocultos de Markov (HMMs) permiten describir de forma estadística el proceso del habla o de la escritura, asumiendo que son procesos estocásticos de Markov, donde los estados están ocultos, y lo que observamos son las emisiones producidas en dichos estados. Los HMMs necesitan de estimadores estadísticos que permitan establecer la probabilidad

de emisión de cada uno de los estados del modelo. Estos estimadores pueden ser combinaciones (mixturas) de Gaussianas en la aproximación convencional, o bien redes neuronales en la aproximación híbrida. Esta última aproximación ha mostrado tener ventajas frente a los HMMs convencionales y por ello será nuestro marco de trabajo.

Los objetivos de este capítulo son:

- Detallar y formalizar la aproximación al reconocimiento estadístico de secuencias con HMMs hibridados con ANNs (HMM/ANN).
- Formalizar el algoritmo de búsqueda para reconocimiento de secuencias con HMMs.

6.1 Descripción

En un modelo de Markov clásico, los estados son directamente visibles para el observador, y por lo tanto el único parámetro son las probabilidades de transición. En cambio, un HMM tiene ocultos los estados, pero son capaces de generar una emisión que sí es visible, cuya distribución de probabilidad depende del estado que la produce, de manera que no todos los estados tienen la misma probabilidad de generar una determinada emisión. Así, la secuencia de emisiones generada por un HMM nos aporta información sobre la secuencia de estados, y recuperando la secuencia de estados se recupera la transcripción textual de la fuente de información.

Podemos encontrar distintos tipos de HMMs, dependiendo del tipo de emisión:

- Discretos: cada estado produce una emisión de un conjunto finito de símbolos.
- Continuos: las emisiones de los estados son puntos en un espacio continuo.
- Semi continuos: una mezcla de las dos anteriores.

En este trabajo sólo utilizaremos HMMs continuos, si bien la siguiente descripción general del problema la haremos en términos de modelos discretos por ser más sencilla. En su versión discreta, un proceso oculto de Markov se puede entender como una generalización del problema de las bolas y las urnas, planteado de esta forma en [Rabiner, 1990]:

> Consideremos un sistema formado por urnas y bolas de colores situado en una habitación. Tenemos un total de N urnas en la habitación, y estas urnas no son observables directamente. Cada urna contiene un número indeterminado de bolas de cada color. En total existen M colores diferentes. El proceso de generación se basa en la existencia de un genio en la habitación que, de acuerdo con un determinado proceso estocástico, escoge una urna inicial. De esta urna, escoge una bola al azar. El color de esta bola se apunta como la primera observación. El genio devuelve la bola a su urna original, sin que en ningún momento podamos saber cual era. El genio selecciona la urna siguiente siguiendo un criterio estocástico de selección que dependerá únicamente de la urna actual, y se repite el proceso de selección de otra bola, así hasta completar toda la secuencia de observaciones.

El proceso es de Markov ya que la elección de la urna depende directamente de la urna anterior. Y es oculto porque el proceso de Markov en sí mismo no es observable, y sólo conocemos la secuencia de bolas extraídas.

Los procesos de reconocimiento automático del habla y reconocimiento de escritura manuscrita pueden ser interpretados como procesos ocultos de Markov, y por tanto se puede utilizar la teoría de los HMMs para modelarlos. En este caso la secuencia de vectores de características es la secuencia de emisiones del HMM, siendo cada vector de características una emisión observada. En el ejemplo de las urnas se asume que después de una urna puede ser elegida cualquier otra. En el caso de las aplicaciones prácticas aquí mencionadas, la estructura del modelo será mucho más restrictiva.

Los HMMs son un tipo de autómata finito estocástico y vienen definidos por una tupla $(\mathcal{Q}, q_F, \Sigma, P_I, P_T, P_E)$, donde:

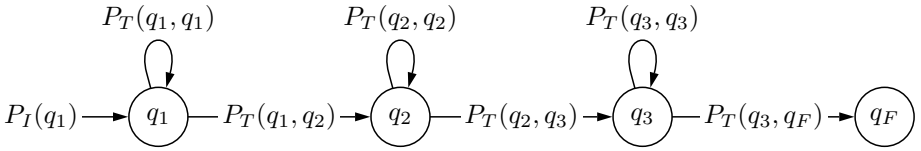


Figura 6.1: *Ejemplo de HMM de tres estados. //* Three states HMM.

- $\mathcal{Q} = \{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$ es el conjunto de estados que define el modelo,
- $q_F \in \mathcal{Q}$ es el estado final del modelo,
- Σ es el conjunto de emisiones del modelo,
- $P_I : \mathcal{Q} \to [0, 1], \sum_{q \in \mathcal{Q}} P_I(q) = 1$ es la distribución de probabilidad de ser inicial,
- $P_T : \mathcal{Q} \times \mathcal{Q} \to [0, 1], \sum_{q' \in \mathcal{Q}} (P_T(q, q') = 1)\,, \forall q \in \mathcal{Q}$ es la probabilidad de transitar de un estado a otro en el modelo,
- $P_E : \mathcal{Q} \times \Sigma \to [0, 1], \sum_{x \in \Sigma} P_E(x|q) = 1 \forall q \in \mathcal{Q}$ es la probabilidad de que un determinado estado del modelo produzca una determinada emisión.

En esta definición hemos asumido modelos de orden 1, es decir, la probabilidad de transitar entre estados únicamente depende del estado anterior.

A lo largo de este trabajo, hemos asumido que existe un único estado inicial q_1, esto es, $P_I(q_1) = 1$. La topología de los modelos es de izquierda a derecha con bucles, es decir, cada estado se conecta con el siguiente y consigo mismo. Cuando transitamos a un estado, este emite siguiendo la distribución P_E. Hay tantos modelos de Markov como símbolos (fonemas en voz, grafemas en escritura) que queremos representar. Se puede ver un ejemplo de topología para el caso de 3 estados en la figura 6.1.

Bajo esta definición, el proceso de búsqueda necesario para obtener la transcripción en un sistema de reconocimiento, ya sea voz o escritura, sigue la ecuación (1.2), que recuperamos aquí:

$$\hat{\bar{y}} = \arg\max_{\bar{y} \in \Omega^+} p(\bar{y}|\bar{x}) = \arg\max_{\bar{y} \in \Omega^+} p(\bar{x}|\bar{y}) p(\bar{y})\,, \tag{1.2}$$

donde:

- Ω es el vocabulario de la tarea;
- $\bar{y} = y_1 \ldots y_{|\bar{y}|}$ es la transcripción de longitud $|\bar{y}|$ ofrecida como salida del sistema;
- $\bar{x} = x_1 \ldots x_{|\bar{x}|}$ es una secuencia de vectores de características de longitud $|\bar{x}|$, entrada del sistema;
- $p(\bar{x}|\bar{y})$ es la probabilidad generativa del HMM;
- y $p(\bar{y})$ es la probabilidad del modelo de lenguaje, que aparece en la ecuación tras la aplicación de la regla de Bayes, ignorando $p(\bar{x})$ por ser una constante en la maximización.

La probabilidad $p(\bar{x}|\bar{y})$, bajo el marco de los modelos de Markov [Rabiner & Huang, 1993], se puede descomponer como:

$$p(\bar{x}|\bar{y}) = \max_{q_1 \dots q_J \in g(\bar{y})} P_I(q_1) \left(\prod_{i=1}^{J-1} P_T(q_i, q_{i+1}) P_E(x_i|q_i) \right) P_E(x_J|q_J)\,, \qquad (6.1)$$

donde $g : \Omega^\star \rightarrow \mathcal{Q}^{|X|}$ es una función que devuelve toda posible secuencia de estados dada la secuencia de palabras $\bar{y}$. La función $P_E(x|q)$ en la aproximación estándar (con HMMs continuos) se estima mediante mixturas de Gaussianas, típicamente una mixtura por cada estado del modelo oculto de Markov. Por simplicidad, denotaremos $P_E(x|q)$ con $p(x|q)$ en el resto del texto.

En la práctica, la ecuación (1.2) se generaliza, introduciendo dos factores que modifican la combinación matemática del HMM y el modelo de lenguaje: el Word Insertion Penalty (WIP) y el Grammar Scale Factor (GSF). Estos parámetros permiten interpretarla como una combinación log-lineal de modelos dentro del marco de Máxima Entropía, siguiendo la ecuación (1.3), como se observa en la ecuación (1.4), que repetimos aquí:

$$\hat{\bar{y}} = \arg \max_{\bar{y} \in \Omega^+} p(\bar{x}|\bar{y}) p(\bar{y})^{\lambda_1} exp(|\bar{y}|)^{\lambda_2}\,, \qquad (1.4)$$

donde λ_1 es el parámetro GSF que se aplica al modelo de lenguaje, y λ_2 es el parámetro WIP que se aplica a un nuevo modelo que sigue una distribución exponencial dependiente del número de palabras de la hipótesis generada y el factor de combinación. El valor de estos nuevos factores se estima mediante el método MERT (véase la sección 1.7), configurado para utilizar el algoritmo Simplex [Nelder & Mead, 1965], que nos permite optimizar los factores λ_i en este tipo de combinaciones log-lineales. Nótese que en la literatura la estimación de estos parámetros tiende a hacerse mediante prueba y error. El MERT está muy extendido en traducción automática, pero en HMMs es *casi* un desconocido.

6.1.1. Los tres problemas básicos de los HMMs

Dada la formalización anterior, son tres los problemas básicos de los HMMs:

1. Problema de la evaluación: dada una secuencia de vectores de características $\bar{x}$, y un modelo $\mathcal{M} = (\mathcal{Q}, q_F, \Sigma, P_I, P_T, P_E)$, ¿Cómo calcular de manera eficiente la probabilidad $P(\bar{x}|\mathcal{M})$? El algoritmo forward-backward [Baum & Egon, 1967; Baum & Sell, 1968] se usará con este cometido.

2. Problema de la decodificación: dada una secuencia de vectores de características $\bar{x}$, y el modelo $\mathcal{M}$, ¿Cómo elegir la secuencia de estados $Q = q_1 q_2 \dots q_{|\bar{x}|}$ óptima (la que mejor explica las observaciones)? El algoritmo de Viterbi [Viterbi, 1967] será utilizado para resolver este problema. Concretamente abordaremos el problema de la decodificación con el algoritmos de dos pasos [Sakoe, 1979]. Se detallará un poco más adelante.

3. Problema del entrenamiento: ¿Cómo ajustar los parámetros del modelo $\mathcal{M}$ para maximizar el valor de $P(\bar{x}|\mathcal{M})$? Para resolver este problema se utilizará una adaptación del algoritmo de Expectación-Maximización (EM) [Dempster *et al.*, 1977] para modelos de Markov hibridados con redes neuronales [España-Boquera *et al.*, 2011].

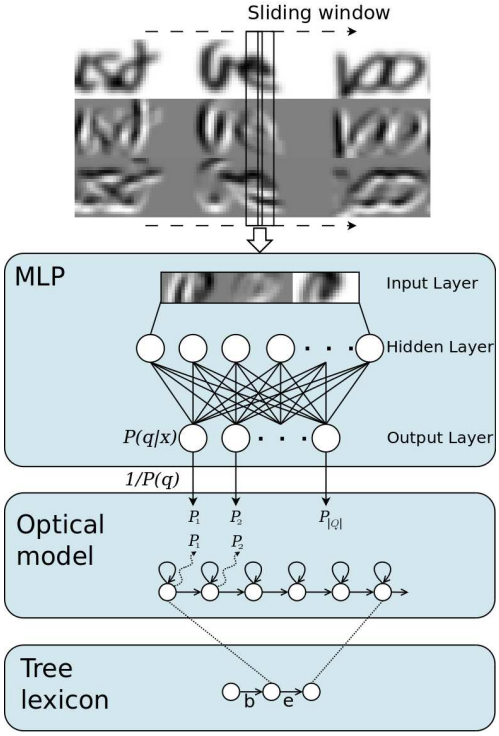


Figura 6.2: *Arquitectura de un sistema HMM/ANN para reconocimiento de escritura manuscrita. //* HMM/ANN architecture for HTR tasks. Referenced at page 305.

6.1.2. Modelos ocultos de Markov hibridados con redes neuronales

El formalismo seguido en esta tesis sustituye las mixturas de Gaussianas por una red neuronal, construyendo lo que se conoce como Modelo Oculto de Markov hibridado con Redes Neuronales Artificiales (HMM/ANN por sus siglas en inglés). De esa manera, en lugar de tener una mixtura de Gaussianas por cada posible emisión del HMM, se tiene una salida de la red neuronal por cada posible emisión. La red neuronal, un perceptrón multicapa, se entrena para aproximar probabilidades a posteriori, $p(q|x)$, asociando cada salida de la red con un estado del modelo oculto de Markov y entrenando la red como si fuera un clasificador [Bourlard & Morgan, 1994; Bishop, 1995]. Se utiliza la función de activación softmax [Bridle, 1990] para normalizar $p(q|x)$ y que la suma de todas las salidas de la red sea uno. La probabilidad a posteriori, $p(q|x)$, se convierte en la probabilidad de emisión, $p(x|q)$, aplicando la regla de Bayes:

$$p(x|q) = \frac{p(q|x)p(x)}{p(q)} \,. \tag{6.2}$$

La probabilidad a priori de las clases, $p(q)$, se puede estimar mediante conteo a partir de la información suministrada por el alineamiento forzado Viterbi durante la etapa de entrenamiento. Por lo tanto, usaremos la verosimilitud escalada $p(x|q)/p(x)$ como probabilidades de emisión en nuestro sistema HMM/ANN. Esto es posible debido a que el factor de escala $p(x)$

es constante para toda posible secuencia de salida durante el reconocimiento, y por tanto, es posible ignorarlo [Bourlard & Morgan, 1994]. De esta manera integramos la red neuronal dentro del sistema tradicional. Se puede ver un esquema de un sistema de reconocimiento de escritura manuscrita en la figura 6.2.

Al tratar la red como un clasificador, es posible añadir diferentes características a la entrada relacionadas con los vectores de características originales. En este trabajo se utilizará siempre una entrada ampliada con un contexto izquierdo y derecho. Para calcular la probabilidad de cada una de las emisiones del HMM, la red neuronal recibirá una serie de tramas adyacentes por la izquierda a la trama actual, y una serie de tramas adyacentes por la derecha, si bien sería posible añadir cualquier característica que pudiera ser interesante para el cálculo de la probabilidad. El algoritmo de entrenamiento está basado en el EM y consiste en ir alternando:

- etapas de segmentación y estimación de HMMs mediante Viterbi (Expectación) con
- etapas de entrenamiento de la red neuronal para que aprenda dicha segmentación (Maximización).

6.2 Arquitectura del algoritmo de decodificación

Ya sabemos que el problema de la decodificación de un HMM se puede resolver mediante el algoritmo de Viterbi. Sin embargo, en aplicaciones prácticas, el HMM por sí sólo no es suficiente. Fijándonos en la ecuación básica del reconocimiento, ecuación (1.2), tras aplicar la regla de Bayes, la fórmula se descompone en el cálculo de dos probabilidades:

- $p(\bar{x}|\bar{y})$ que se modeliza mediante el marco de HMMs descrito en este capítulo,
- $p(\bar{y})$ que se modeliza mediante un modelo de lenguaje, visto en el capítulo 2.

Este problema, más general que el de los HMMs, precisa desarrollar una adaptación del algoritmo de Viterbi que permita añadir el cálculo del modelo de lenguaje en el proceso de decodificación.

Otra solución es utilizar el algoritmo de dos pasos [Sakoe, 1979], modificado para permitir que ambos pasos puedan ser ejecutados de forma síncrona. Además, utilizar un algoritmo de dos pasos permite aplicar de forma muy simple técnicas de sobresegmentación que pueden permitir mejorar los resultados especialmente en reconocimiento de escritura. Este algoritmo es el fruto del desarrollo conjunto dentro del grupo de investigación. Será expuesto aquí, aunque es parte de la tesis de Salvador España [España-Boquera, 2012].

6.2.1. Sobresegmentación

En un algoritmo de dos pasos estándar se asume que cualquier posición de la secuencia de vectores de características puede ser lo que denominamos *frontera*, esto es, un punto donde puede terminar y empezar una palabra. Debido a esto, el coste computacional del algoritmo es muy elevado siendo necesario aplicar una poda estática (o dinámica) muy fuerte.

El algoritmo de dos pasos utilizado permite dar información de fronteras a los algoritmos de decodificación, de esa manera se restringe la búsqueda. Las fronteras se clasifican en:

- fronteras interpalabra: aquellas que separan palabras;

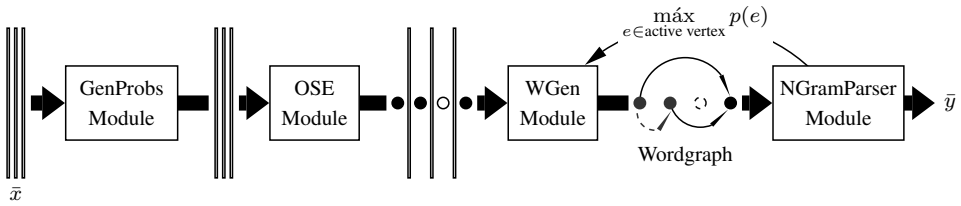


Figura 6.3: *Diagrama de bloques de la arquitectura* `dataflow` *para reconocimiento de secuencias mediante HMMs. El módulo FrameSource no ha sido representado, el vector $\bar{x}$ es la salida de dicho módulo. //* `Dataflow` diagram for HMM decoding. Referenced at page 305.

- fronteras intrapalabra: son fronteras que separan tramas pertenecientes a una misma palabra.

Las fronteras van siempre entre dos vectores de características, excepto la primera y la última, que van precediendo y sucediendo al primer y último vector de características respectivamente. Esto significa que siempre tendremos una frontera más que número de vectores. Las fronteras son valores booleanos indicando si la frontera es o no interpalabra: un 1 significa que la frontera es interpalabra, un 0 significa que es intrapalabra.

Utilizaremos técnicas para permitir sobresegmentar de forma *inteligente* en el reconocimiento de escritura manuscrita. En reconocimiento de voz esta tarea es mucho más compleja y no ha sido abordada en este trabajo.

6.2.2. Módulos dataflow

El algoritmo se ha descompuesto en una serie de módulos de la arquitectura `dataflow`, que se conectan entre sí intercambiando información. Puede verse en la figura 6.3 la arquitectura de conexiones del sistema. Los módulos son los siguientes:

- Fuente de vectores de características: **FrameSource**.
- Calculo de la probabilidad generativa del HMM ($p(x|q)\,\forall q$): **GenProbs**.
- Sobresegmentador, genera las fronteras entre tramas: **OSE**.
- Generador de grafo de palabras a partir de las tramas y las fronteras: **WGen**.
- Viterbi de N-gramas, consume el grafo de palabras y aplica el modelo de lenguaje ($p(\bar{y})$): **NGramParser**.

Módulo FrameSource

Este módulo se encarga de introducir en el sistema los vectores de características extraídos de la señal. El módulo es intercambiable, de manera que permite generar vectores de características a partir de la lectura directa de micrófono, o bien generar las tramas leyendo de fichero, de un socket, ...

Los vectores de características son generados siguiendo el proceso de parametrización requerido (según se trate de voz o imágenes de texto escrito).

Módulo GenProbs

Este módulo recibe los vectores de características del modulo anterior y genera a su salida, por cada vector de características de entrada, un vector con la probabilidad de que cada estado del HMM emita el vector de características correspondiente. Por tanto la salida de este módulo es un vector de la talla del número de emisiones posibles en el HMM.

Este módulo permite cambiar su modo de funcionamiento dependiendo del tipo de HMM que se esté utilizando:

- Si se trata de un HMM/ANN, este módulo utilizará una red neuronal para calcular las probabilidades a posteriori de las emisiones, y después aplicará la división por las probabilidades a priori de las emisiones, para calcular a través de la regla de Bayes la probabilidad generativa que será utilizada en el modelo de Markov, tal y como se ha explicado en la sección 6.1.2.

- Si se trata de un HMM estándar, las probabilidades generativas se calculan mediante mixturas de Gaussianas, de manera que este módulo se encargará de calcular todas las probabilidades posibles.

En todos los experimentos de esta tesis se han utilizado modelos híbridos HMM/ANNs y el funcionamiento de este módulo se ha optimizado para estos modelos: las redes neuronales calculan la probabilidad de *todas* las emisiones en un mismo paso forward aunque sólo se vayan a necesitar unas pocas.

Módulo OSE

Es el módulo encargado de la sobresegmentación. Vamos a distinguir tres tipos básicos de sobresegmentadores:

- Dummy: pone las fronteras siempre marcadas como fronteras interpalabra, equivalente a que no exista la sobresegmentación. Este sobresegmentador se ha utilizado para las tareas de reconocimiento del habla (en el capítulo 7) y de reconocimiento de escritura manuscrita con modelos de lenguaje de grafemas (en la sección 8.8).

- Tabla: recibe las fronteras en una tabla, de manera que no es utilizable en sistemas on-line. Dicha tabla puede ser generada mediante cualquier algoritmo apropiado para ello. Este es el tipo que usaremos en reconocimiento de escritura manuscrita, con modelos de lenguaje basados en palabras. La técnica utilizada para detectar las fronteras está basada en componentes conexas de la imagen del texto manuscrito.

- OnLine: va generando la sobresegmentación a medida que van llegando los vectores de probabilidades generativas. Este tipo de sobresegmentador está implementado para escritura manuscrita on-line, donde es posible emitir fronteras únicamente en los cambios de trazo, cuando el usuario levante el lápiz óptico.

Sólo usaremos los dos primeros tipos de sobresegmentación en este trabajo. El módulo recibirá a su entrada la secuencia de vectores con las probabilidades generativas, y generará a su salida una secuencia donde se alternarán vectores de probabilidades generativas con fronteras, empezando y terminando siempre con fronteras.

Módulo WGen

Este módulo recibe los vectores de probabilidades generativas, más la información de sobresegmentación, y se encarga de generar un grafo de palabras en el formato del protocolo de incidencia descrito en la sección 1.4.1. Es el módulo encargado de la primera etapa del algoritmo de dos pasos. Utiliza un algoritmo de Viterbi basado en léxico en forma de árbol [España-Boquera *et al.*, 2007b], donde dicho árbol es la expansión de un diccionario de palabras formadas por HMMs. El grafo de palabras generado asocia a cada palabra la probabilidad del HMM en el segmento de señal de entrada correspondiente a su arista. Por tanto, recibe la probabilidad generativa como entrada y la multiplica por las probabilidades de transitar P_T y de ser inicial P_I que correspondan.

El funcionamiento del módulo consiste en disponer de un "pool" de instancias del algoritmo de Viterbi para léxico en árbol (de ahora en adelante ViterbiM), donde cada instancia contiene una listas de estados activos. Cada instancia de ViterbiM está identificada por el número de frontera donde fue creada (clave de una tabla hash) y tiene asociada la siguiente información (valor):

- Lista de estados activos, donde cada estado guarda su posición en el léxico en árbol y la probabilidad acumulada.
- Puntero al léxico en árbol.
- Número de frontera donde se creó la instancia, dicho número representa un vértice en el grafo de salida.
- Puntuación del mejor estado activo asociado al vértice donde se creó esta instancia en el siguiente módulo `dataflow`. Esto sirve para permitir que se puede aplicar una poda con criterio global, si bien esta probabilidad no es tenida en cuenta al emitir los arcos del grafo de palabras.

El algoritmo funciona de forma síncrona en el tiempo, con lo que en un mismo instante todos los estados activos habrán consumido exactamente el mismo número de tramas.

El módulo responde a la entrada, según el tipo de dato que recibe. Consecuentemente reacciona con tres acciones diferentes:

- Llega una frontera interpalabra no activa. No hace nada.
- Llega una frontera interpalabra activa (véase algoritmo 6.1). El algoritmo genera un nuevo vértice en el grafo de salida, identificado por el número de la frontera. Después pide a las instancias de ViterbiM las palabras que pueden emitirse en dicha frontera[1]. Con la información de los estados que son final de palabra, y la frontera en la que empezó cada una de las instancias, es posible generar arcos que inciden en el vértice recién generado, y que salen del vértice asociado al número de frontera inicial de cada instancia. Seguidamente se envía la señal de que no van a llegar más aristas al nuevo vértice (`no_more_in_edges`).

  Tras enviar el `no_more_in_edges` el algoritmo queda a la espera de que el siguiente módulo `dataflow` le conteste con el mensaje `score_feed_back`, que contiene la

[1] Una palabra se emite cuando un estado activo se encuentra en un nodo del léxico en árbol que es final de palabra.

mejor probabilidad del sistema de reconocimiento habiendo llegado hasta el vértice activo. Esta probabilidad incluye a todos los modelos de la combinación (HMM, modelo de lenguaje, penalización por número de palabras), y sirve para crear una nueva instancia de ViterbiM en el pool. Dicha instancia se asocia al número de frontera actual (vértice emitido), y se guarda la probabilidad recibida en el mensaje `score_feed_back` para poder utilizarla en la poda.

- Llega un vector de probabilidades de emisión (véase algoritmo 6.2). El módulo recorre el pool y aplica una etapa a cada instancia del algoritmo ViterbiM contenida en él. Se utiliza poda en haz y poda estática para permitir acelerar el cálculo. Si una lista de estados activos se queda vacía, es eliminada del pool. La poda es aplicada de forma global a todos los estados activos de todas las instancias de ViterbiM en el pool, ya que todos han consumido exactamente el mismo número de tramas.

Algoritmo 6.1 Genera los mensajes `dataflow` para un vértice y los arcos que inciden en él, para generar el grafo de palabras del primer paso del algoritmo de reconocimiento. // *Generation of* `dataflow` *messages for a vertex and its incoming edges. Referenced at page 306.*

Require: A ViterbiM pool of parsers
Ensure: Generate all edges that will income to the graph vertex that begins at the current frontier

```
1:  function InterWordFrontier()
2:  begin
3:    Let v = new vertex index for current frontier
4:    Generate message vertex (v)
5:    for all parser p ∈ ViterbiM pool do
6:      Let u = initial vertex index associated to parser p
7:      Let L = ∅ an empty list of pairs (word,score)
8:      for all possible pairs ⟨w, s⟩ of (word,score) generated by parser p do
9:        Update s = s/initial mass of parser p
10:       Append ⟨w, s⟩ to L
11:     end for
12:     Generate message edge (u, data=L) // edge from u to v
13:   end for
14:   Generate message no_more_in_edges (v)
15:   Wait for message score_feed_back and store its returned value at bestscore
16:   Put on the pool a new ViterbiM parser instance using bestscore as initial mass,
      bestscore as the score of the first active state, and v as initial vertex index
17: end function
```

Módulo NGramParser

Este es el módulo encargado de aplicar Viterbi con el modelo de lenguaje de N-gramas. Este algoritmo recibe a su entrada un grafo serializado siguiendo el protocolo de incidencia, y va realizando acciones en función del mensaje que le llega.

Algoritmo 6.2 Aplica un paso de ViterbiM con una trama. *// It applies a ViterbiM step with one frame. Referenced at page 306.*

Require: An input frame f_i formed by all emission probabilities $P_E(x_i|q)$ of each HMM state q, a ViterbiM pool of parsers

Ensure: Compute a ViterbiM step on each parser and prune the worst active states using the same pruning criteria

1: **function** ApplyFrame(f_i)
2: **begin**
3: Let bestscore $= 0$
4: **for all** parser $p \in$ ViterbiM pool **do**
5: Compute Viterbi step with all active states of parser p using frame f_i
6: Update bestscore $= \max($bestscore, best active state score at $p)$
7: **end for**
8: Prune worst active states at parsers pool using beam width (based on bestscore) and histogram pruning
9: **end function**

El módulo funciona guardando listas de estados activos asociadas a los vértices del grafo que recibe a su entrada. Cada una de estas listas tiene un tamaño máximo para aplicar así poda estática. Cada estado activo contiene:

- Estado en el modelo de lenguaje de N-gramas. En caso de ser un modelo de N-gramas estándar, este estado se representa por un índice al autómata que guarda el modelo. En el caso de NN LMs, este estado es un LMkey al TrieLM (véase el capítulo 2).
- Probabilidad acumulada de llegar hasta dicho estado activo partiendo desde el estado activo inicial en el vértice inicial del grafo de palabras.
- Backpointer al estado activo desde el cual llegamos a este con dicha probabilidad.

Cada estado activo se encuentra en una lista asociada a cada vértice del grafo de entrada. Para cada vértice del grafo guardamos:

- Lista de estados activos.
- Probabilidad del mejor de los estados activos en la lista.
- Identificador del vértice.

El módulo responde a los siguientes mensajes del protocolo de incidencia:

- `vertex` (id): al llegar un mensaje de este tipo el algoritmo crea un nuevo vértice, le asocia una lista de estados activos vacía y marca el id del vértice recibido como vértice activo.
- `edge` (id, data=$\{\langle w_1, s_1\rangle, \langle w_2, s_2\rangle, \ldots\}$) (véase algoritmo 6.3): al llegar una arista con sus correspondientes palabras y probabilidades, se recorre la lista de estados activos del vértice origen id y se transita en el modelo de lenguaje con cada una de las palabras de data, aplicando la probabilidad de modelo de lenguaje usando GSF como factor de

combinación, la correspondiente s_i y el WIP. La lista de estados activos del vértice destino se actualiza convenientemente maximizando la probabilidad en caso de que ya hubiera un estado activo con el mismo LMkey en el modelo de lenguaje.

- `no_more_in_edges` (id): cuando llega este mensaje se envía al módulo anterior el mensaje `score_feed_back` con la probabilidad del mejor estado activo en el vértice activo. En este mismo instante se aprovecha para podar la lista de estados activos asociada al vértice activo.

- `is_initial` (p): crea en el último `vertex` recibido un estado activo inicial usando el LMkey devuelto por getInitialLMkey, y con probabilidad p.

- `is_final` (p): transita con el modelo de lenguaje en los estados activos del vértice activo, aplicando la transición al LMkey devuelto por getFinalLMkey, y transitando a un nuevo estado ficticio del grafo de entrada que funciona como vértice final. Aplica la probabilidad p a cada estado activo resultante de la transición en el modelo de lenguaje.

Algoritmo 6.3 Método `edge` del reconocedor basado en HMMs. // `Edge` *method. Referenced at page 306.*

Require: Origin vertex u, current vertex v, and (word,score) pairs list.
Ensure: Apply a Viterbi step using the language model score for active states at u using the words at the pairs list, the HMM score at the pairs list, and the WIP

1: **function** Edge(u, data $= \{\langle w_1, s_1\rangle, \langle w_2, s_2\rangle, \ldots\}$)
2: **begin**
3: **for all** active state $a \in$ Vertex(u) **do**
4: Let $k =$ LMkey at a
5: Call method prepareLM(k)
6: **for all** pairs $\langle w_i, s_i\rangle$ of (word,score) $\in$ data **do**
7: Let $p = s_i \cdot$ getLMprob$(k, w_i)^{\text{GSF}} \cdot$ WIP
8: Let $k' =$ getNextLMkey(k, w_i)
9: Let $a' =$ active state with LMkey k' at Vertex(v) // v is the current active vertex
10: **if** a' score $< p$ **then**
11: Update a' score $= p$
12: Update a' backpointer $= a$
13: **end if**
14: **end for**
15: **end for**
16: **end function** Edge

Cuando termina el proceso de decodificación, es posible solicitar estos tres procedimientos al módulo:

- getBestPath: devuelve el mejor camino, comenzando en un estado final y llegando a un estado inicial. Escoge siempre el estado activo con mayor probabilidad, y va tirando hacia detrás por sus backpointers.

- getNextHypotesis: devuelve la siguiente mejor hipótesis. Esto permite recibir una lista de N-best. La lista de N-best se genera mediante sucesivas llamadas a este método.

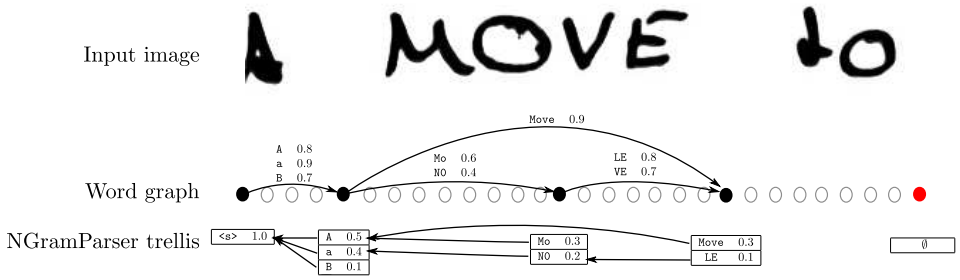


Figura 6.4: *Ejemplo de decodificación incompleta mediante HMMs. //* Incomplete decoding example with HMMs. Referenced at page 306.

Implementa el algoritmo de Eppstein [Eppstein, 1998] para los K mejores caminos en problemas de programación dinámica, concretamente implementa su versión Lazy [Jiménez & Marzal, 2003].

- getOutputWordGraph: devuelve un grafo que contiene las N mejores hipótesis, de nuevo generado a través del algoritmo de Eppstein [Eppstein, 1998; Jiménez & Marzal, 2003].

La figura 6.4 muestra un ejemplo de decodificación sin terminar. Podemos ver una imagen alineada con los vértices de un grafo de ejemplo del primer paso del algoritmo. En rojo está el vértice actual, que todavía no ha recibido aristas. Cada vértice tiene alineado en la parte inferior la lista de estados activos del algoritmo del segundo paso en el caso de que el modelo de lenguaje fuera de bigramas, y los backpointers que conforman el trellis del algoritmo de Viterbi. Comenzando por el final, y siguiendo la dirección de los backpointers, podemos extraer la mejor hipótesis resultado de la decodificación. Para poder aplicar el algoritmo de Eppstein y obtener listas o grafos con las N-best, es necesario en lugar de guardarse el mejor backpointer, guardarse los H mejores, de manera que a mayor valor de H, mejores hipótesis aparecen en la lista de las N-best. [2]

6.3 Resumen

Este capítulo ha presentado la aproximación estadística al reconocimiento de secuencias a través de los modelos ocultos de Markov, junto a su formalización matemática y algorítmica, exponiendo los detalles más importantes del algoritmo de búsqueda implementado para el desarrollo de esta tesis. Dicho algoritmo es uno de los resultados de la tesis de Salvador España [España-Boquera, 2012], y la implementación de los mismos es parte del trabajo conjunto necesario para llevar a cabo la experimentación de nuestras respectivas líneas de investigación.

[2] La "one-best" sigue siendo la misma, pero aparece una mayor proporción de hipótesis con probabilidad cercana a la one-best.

7

RECONOCIMIENTO Y COMPRENSIÓN AUTOMÁTICA DEL HABLA

Índice

El reconocimiento automático del habla o "Automatic Speech Recognition" (ASR) es uno de los campos más amplios y antiguos del área del reconocimiento de formas e inteligencia artificial. El lenguaje hablado es la forma de comunicación más importante para los seres humanos, y por ello desde la existencia de las primeras máquinas hemos deseado esta forma de comunicación con ellas. La aproximación estándar al ASR se basa en utilizar HMMs para modelizar la voz, asumiendo que es un proceso estocástico. Se entrenan HMMs y modelos de lenguaje, y se decodifica utilizando el algoritmo de Viterbi [Viterbi, 1967]. Uno de los retos más interesantes de las tecnologías del habla es la comprensión del lenguaje hablado o "Spoken Language Understanding" (SLU). El SLU consiste en traducir el lenguaje hablado a un lenguaje conceptual que facilita su procesamiento automático. Esta tarea es el primer paso en un sistema de diálogo automático.

En este capítulo introduciremos los conceptos más importantes del ASR, así como una introducción a la comprensión del lenguaje. La incorporación de NN LMs a tareas de comprensión en sistemas de diálogo hablado usará los Cache NN LMs descritos en la sección 5.3.

Los objetivos de este capítulo son:

- Introducción al ASR, y concretamente, a la SLU.
- Diseño de un sistema baseline de ASR, con prestaciones competitivas, similares a las del estado del arte.
- Extensión del sistema baseline para su uso en tareas de SLU, modelando la tarea de comprensión como una tarea de etiquetado mediante modelos de lenguaje especializados.
- Mejora del sistema baseline de SLU aplicando la técnica de los Cache NN LMs a los modelos de lenguaje utilizados. Esto permitirá añadir la historia del diálogo al modelo de lenguaje para adaptar la distribución de probabilidad del modelo al contexto.

7.1 Introducción

El ASR basa su formalización en los HMMs. Para ello se asume que la voz es un proceso estadístico generado a través de un proceso oculto de Markov. Bajo esta hipótesis, las observaciones del HMM son los sonidos que producimos al hablar, si tomamos el sonido como instantes de corta duración (por ejemplo 10 milisegundos). Estas observaciones están asociadas a los estados del modelo, de manera que cada observación tiene una probabilidad de haber sido generada en un determinado estado. Los estados representan fonemas, o bien partes de un fonema si usamos modelos de 3 estados para cada fonema: el comienzo del fonema, la parte central, y su final. Las transiciones indican la probabilidad de cambiar el aparato fonador de una configuración determinada a otra.

A partir de la ecuación general del reconocimiento, ecuación (1.2):

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{y}|\bar{x}) = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y}) \qquad (1.2)$$

y tras aplicar la regla de Bayes, obtenemos el término $p(\bar{x}|\bar{y})$ que se conoce como modelo acústico, donde $\bar{x}$ es la secuencia de observaciones, sonidos en este caso, e $\bar{y}$ es la secuencia de palabras que estamos evaluando como hipótesis. Como ya se explicó en el capítulo 6, el sistema está formado por el modelo acústico que calcula la probabilidad condicional $p(\bar{x}|\bar{y})$ y por el modelo de lenguaje que calcula la probabilidad a priori $p(\bar{y})$ de la secuencia de palabras. Siguiendo la ecuación (1.3) se introducen los parámetros GSF (λ_1) y WIP (λ_2) en la combinación de ambos modelos. La ecuación (1.4) sigue esta aproximación, y la retomamos aquí:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y})^{\lambda_1} exp(|\bar{y}|)^{\lambda_2} \,. \qquad (1.4)$$

Por tanto, el modelo de lenguaje es una pieza clave en tareas de ASR y nos va a permitir dotar al sistema de la flexibilidad suficiente como para llevar a cabo SLU sin cambiar *casi* el paradigma de trabajo. Fue en el campo del ASR donde los NN LMs comenzaron a utilizarse obteniendo buenos y prometedores resultados [Schwenk & Gauvain, 2002, 2005; Schwenk, 2007]. Aquí los extenderemos a SLU, y evaluaremos la idea de los modelos tipo caché comparando el efecto que tiene utilizar más o menos información en el módulo caché.

7.2 Preproceso y parametrización de la voz

La voz es una onda, y como tal se representa como una función que nos indica la intensidad acústica en cada instante[1]. Para poder llevar a cabo la modelización mediante HMMs es necesario convertir esa función de intensidad en una secuencia de vectores de características. Para obtener la secuencia $\bar{x}$ de vectores de características necesaria para nuestro sistema hay que realizar varios pasos:

- adquirir la señal de voz;
- preprocesar la señal, para eliminar ruido y realzarla, consiguiendo una mejor escucha;
- parametrizar la señal, para hacer que sea menos redundante (más compacta), realzando de nuevo la parte importante.

[1] Toda esta sección es un resumen de [Marzal, 2006].

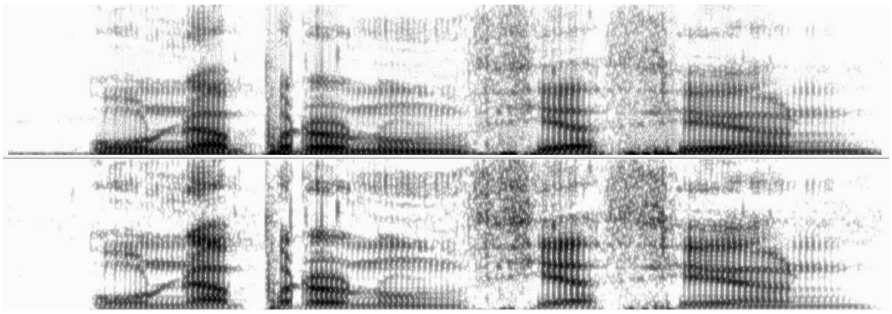


Figura 7.1: *Sonograma correspondiente a* una pronunciación*, Arriba: sin preénfasis, Abajo: con preénfasis (factor* $a = 0.97$*). En cada una de las imágenes, las frecuencias altas están en la parte superior de la misma. (*Figura extraída de [Marzal, 2006]*). //* An example of a sonogram.

7.2.1. Adquisición de la señal

El proceso de adquisición de voz se realiza normalmente mediante un micrófono. Este dispositivo transforma la presión que recibe del aire en una señal eléctrica. Una vez la señal ha sido capturada, esta es digitalizada para su procesamiento automático en un ordenador. Para digitalizar la señal se discretiza a una determinada *frecuencia* de muestreo (F_s) y a unos determinados niveles de *cuantificación* o resolución. La frecuencia se mide en Hz y la cuantificación en bits.

La mayor parte de la energía del habla se encuentra por debajo de 7 kHz, con lo que, siguiendo el teorema de Nyquist, $F_s = 16$ kHz es una frecuencia de muestreo más que suficiente para una aplicación de voz. No obstante, si la voz se obtiene de una línea telefónica, basta con muestrear a $F_s = 8$ kHz, ya que la línea telefónica tiene su energía repartida entre los 300 y 3400 Hz.

En cuanto a la cuantificación, el rango dinámico del oído humano se encuentra en torno a los 20 bits, si bien 12 bits es suficiente para recoger el rango dinámico de la señal oral. Por tanto, los sistemas de reconocimiento de voz suelen trabajar a 12 bits, si bien estos son codificados en 8 bits utilizando una función de compresión no lineal.

7.2.2. Preproceso de la señal

Preénfasis

En el dominio de la frecuencia las características de la voz se manifiestan como *formantes*, que son picos debidos a la resonancia del tracto vocal. Los formantes de alta frecuencia tienen una amplitud menor que los formantes de baja frecuencia, sin embargo la información que aportan es muy importante.

El proceso de preénfasis pretende realzar las altas frecuencias y así no aumentar su resolución. Este proceso consiste en un simple filtro de respuesta finita o "Finite Impulse Response" (FIR), cuya función de transferencia en el dominio z es:

$$H(z) = 1 - az^{-1}\,, \qquad 0 \leq a \leq 1\,,$$

donde a es el parámetro del preénfasis. El cálculo a efectuar es muy sencillo y sigue la siguiente ecuación:

$$s'_n = s_n - a s_{n-1}\,,$$

donde s_n es la n-ésima muestra de la señal original, y s'_n la n-ésima muestra de la señal preenfatizada. Típicamente se utiliza un valor de $a = 0.95$ o $a = 0.97$, aplicando una amplificación de más de 20 dB para las altas frecuencias. La figura 7.1 ilustra un ejemplo de la señal de voz en el dominio de las frecuencias antes y después de aplicar el preénfasis.

Otros preprocesamientos

Según el entorno de trabajo otros tipos de preprocesamientos pueden ser deseables, como son:

- la cancelación de ruidos, por ejemplo el ruido ambiente, ruido de motores, ruido inducido por instalaciones eléctricas, ...
- eliminación de componentes continuas en la señal;
- ...

7.2.3. Parametrización

Análisis en bloques

Debido a las inercias del aparato articulatorio humano, las variaciones en la señal se producen cada 80-200 milisegundos, de manera que cada *región estable* quedará descrita por unas 100-300 muestras. La señal se *trocea* en bloques contiguos con cierto solapamiento. A la frecuencia con que obtenemos los bloques se le denomina *frecuencia de submuestreo*, o de manera equivalente, es el avance de la ventana de submuestreo. El tamaño de dicha ventana de submuestreo es tal que permita el solapamiento entre las ventanas. Con este proceso se convierte la señal en una *secuencia de vectores* ("frames"). La figura 7.2 ilustra un ejemplo de este procedimiento.

Análisis en frecuencias

Una vez la señal ha sido segmentada en bloques, se procede a analizar la energía de cada frecuencia de audio. El resultado de este análisis son los *sonogramas*, como los vistos en la figura 7.1. Se convierte cada vector (correspondiente a cada bloque) en un nuevo vector donde cada componente nos indica la energía de una determinada frecuencia. Este procedimiento se lleva acabo aplicando la transformada discreta de Fourier (FFT) a cada bloque.

Ventana de suavizado espectral

Al segmentar en bloques la señal, aparecen saltos muy abruptos en las frecuencias debidos al inicio y al fin de cada bloque. Para evitar que este ruido perjudique al sistema de reconocimiento, se aplica una familia de filtros, conocidos como *ventanas de Hamming*.

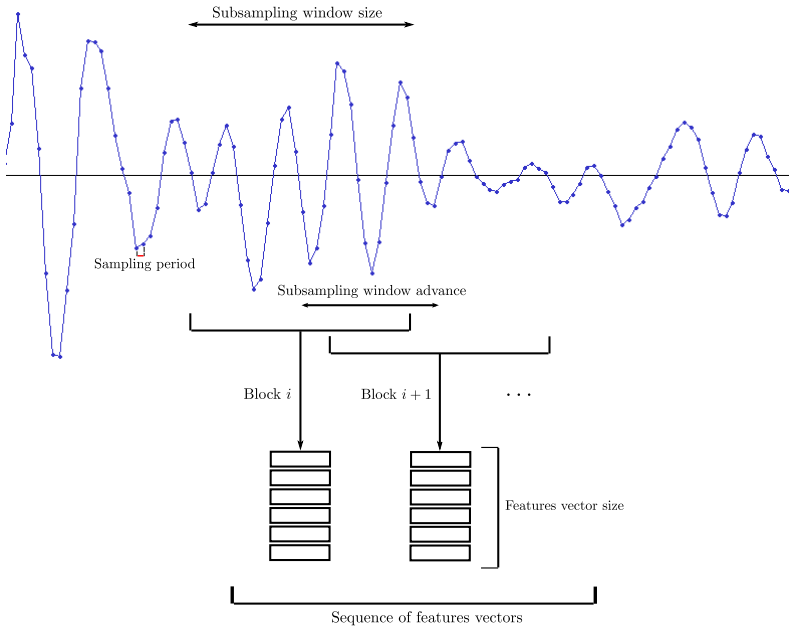


Figura 7.2: *Fragmentación en bloques de la señal oral. (*Figura extraída de [Marzal, 2006]*). //* Segmentation into blocks of a speech signal.

Banco de filtros

El banco de filtros permite filtrar la salida de la FFT para que la energía tenga una distribución no uniforme. La escala de Mel permite obtener la energía de cada banda de frecuencias como la combinación ponderada de las bandas adyacentes en la FFT, asemejándose a la forma en que el oído humano reparte la energía de las frecuencias escuchadas (véase la figura 7.3).

Decorrelación del banco de filtros

La salida del banco de filtros son valores fuertemente correlacionados debido a la convolución aplicada por los filtros. Esto suele ser una fuente de problemas cuando se trabaja en sistemas estadísticos. Hay diversas formas de decorrelacionar la salida del banco de filtros:

- Coeficientes cepstrales: se aplica la transformada inversa sobre el logaritmo de la salida del banco de filtros. Esta es la aproximación estándar en reconocimiento automático del habla, y genera lo que se conoce como "Mel Frequency Cepstrum Coefficients" (MFCCs).
- Otros autores [Nadeu *et al.*, 1996] proponen transformaciones más eficientes de calcular, que permiten utilizar directamente el banco de filtros, ya que esté se produce decorrelacionado.

En este trabajo utilizaremos la aproximación de [Nadeu *et al.*, 1996] que permite construir el banco de filtros decorrelacionado sin calcular coeficientes cepstrales, y además promete obtener buenos resultados en entornos ruidosos.

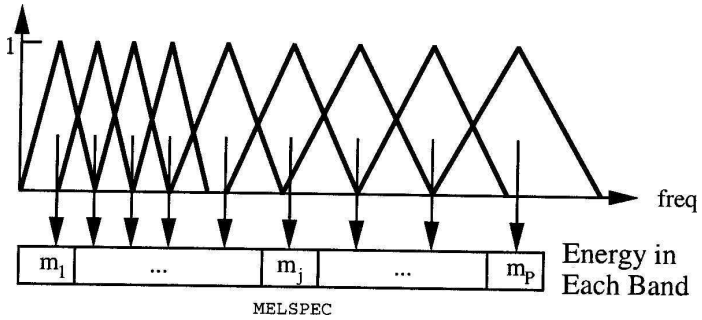


Figura 7.3: *Ejemplo de la escala de Mel. (*Figura extraída de [Marzal, 2006]*). //* Mel-scale example.

7.3 Corpus French Media

La experimentación de este capítulo se ha realizado utilizando la parte francesa del corpus Media [Bonneau-Maynard *et al.*, 2005], disponible en el repositorio de recursos lingüísticos ELDA. Consta de grabaciones de diálogos telefónicos que han sido transcritos manualmente y anotados conceptualmente. El corpus consta de 1 250 diálogos grabados por 250 locutores diferentes. Existen ocho categorías de escenarios, cada una con un nivel de complejidad diferente. Fue obtenido a través de la estrategia del *Mago de Oz*, donde un humano simulaba la acción de un servicio telefónico automático de atención turística.

El corpus está dividido en tres partes:

- Entrenamiento, que consta de 12 811 frases, 87 297 palabras y 42 251 conceptos.
- Validación, formada por 1 241 frases, 9 996 palabras y 4 652 conceptos.
- Test, compuesto de 3 468 frases, 24 598 palabras y 11 790 conceptos.

El vocabulario de la tarea, extraído de la partición de entrenamiento, consta de $\Omega = 2\,007$ palabras y $\Omega_C = 72$ conceptos diferentes. La figura 7.4 ilustra un fragmento de una interacción del corpus French Media. La figura 7.5 ilustra el mismo fragmento pero con las etiquetas conceptuales, soportes de los conceptos y su valor correspondiente.

7.4 Entrenamiento de los modelos acústicos

El entrenamiento de los modelos acústicos se ha realizado con la partición de entrenamiento, que asciende a un total de 11 horas de grabación correspondientes a turnos de usuarios, grabados a 8kHz. Adicionalmente se han utilizado 39 horas de habla extraídas del corpus ESTER-II [Galliano *et al.*, 2005], correspondientes a grabaciones de conversaciones telefónicas. El factor de preénfasis se ha fijado a 0.95. Los datos acústicos han sido parametrizados utilizando [Nadeu *et al.*, 1996] con un tamaño de ventana de 30ms y un avance de 10ms, y 12 bancos de filtros en la escala de Mel, decorrelacionados usando la técnica FF1 de [Nadeu *et al.*, 1996]. Los vectores de características obtenidos se normalizan sustrayendo la media y dividiendo por la desviación típica de cada componente de los vectores. Las transcripciones fonéticas utilizadas son las de [Linarès *et al.*, 2007] y los modelos acústicos son

WOZ: *Bienvenue sur le serveur MEDIA système d'informations touristiques et de réservation d'hôtel. Souhaitez vous écouter le guide.*

SPK: Oui

WOZ: *Ce serveur vocal vous permet d'obtenir des informations touristiques sur les principales villes de France et d'effectuer des réservations d'hôtel. Ce service fonctionne avec de la reconnaissance automatique de la parole. Parlez naturellement quelle information voulez vous obtenir.*

SPK: Une réservation d'hôtel.

WOZ: *Dans quelle ville souhaitez vous réserver un hôtel?*

SPK: À Marseille.

WOZ: *Vous souhaitez réserver à Marseille?*

SPK: La troisième semaine d'août.

WOZ: *Vous souhaitez réserver à Marseille la troisième semaine d'août dans quel quartier de Marseille. Souhaitez vous réserver?*

SPK: C'est sans importance.

WOZ: *Merci de patienter. Je recherche vos informations. Vous souhaitez réserver à Marseille pour la troisième semaine d'août. Je vous propose trois hôtels: le Sofitel palais des congrès, hôtel de la Gardanne, Mercure Saint Charles. Souhaitez vous réserver dans l'un de ces hôtels ou obtenir des informations supplémentaires?*

SPK: Alors j'ai des informations supplémentaires, Je voudrais une chambre pour un couple, deux adultes, moins de deux cents euros.

WOZ: *Merci de patienter. Je recherche cette information. À l'hôtel Sofitel palais des congrès la chambre double est à deux cent cinquante euros. À l' hôtel de la Gardanne la chambre double est à cent vingt cinq euros. Au Mercure Saint Charles la chambre double est à soixante quinze euros. Souhaitez vous réserver dans l'un de ces hôtels ou obtenir d'autres informations?*

SPK: Réserver dans l'un de ces hôtels.

WOZ: *Vous souhaitez réserver dans quel hôtel?*

SPK: Gardanne.

WOZ: *Merci de patienter. Je vérifie les disponibilités à l'hôtel Gardanne. Une chambre est disponible correspondant à vos critères la chambre est à cent vingt cinq euros. Est ce que je vous réserve cette chambre?*
...

Figura 7.4: *Fragmento del texto de una interacción usuario-máquina del corpus French Media. En cursiva el texto que corresponde a la respuesta de la máquina. //* Fragment of an user-machine interaction from the French Media corpus. Referenced at page 306.

```
365_1_woz @woz{bienvenue sur le serveur MEDIA système d' informations touristiques et
              de réservation d' hôtel souhaitez vous écouter le guide}[]

365_2_spk @reponse{oui}[oui]

365_3_woz @woz{ce serveur vocal vous permet d' obtenir des informations touristiques
              sur les principales villes de France et d' effectuer des réservations
              d' hôtel ce service fonctionne avec de la reconnaissance automatique
              de la parole parlez naturellement quelle information voulez vous
              obtenir}[]

365_4_spk @command-tache{une réservation}[reservation]
          @objetBD{d' hôtel}[hotel]

365_5_woz @woz{dans quelle ville souhaitez vous réserver un hôtel}[]

365_6_spk @localisation-ville{à Marseille}[marseille]

365_7_woz @woz{vous souhaitez réserver à Marseille}[]

365_8_spk @rang-temps{la troisième}[troisieme]
          @temps-unite{semaine}[semaine]
          @temps-mois{d' août}[8]

365_9_woz @woz{vous souhaitez réserver à Marseille la troisième
              semaine d' août dans quel quartier de Marseille
              souhaitez vous réserver}[]

365_10_spk @reponse{c' est sans importance}[jeSaisPas]

365_11_woz @woz{merci de patienter je recherche vos informations vous souhaitez
               réserver à Marseille pour la troisième semaine d' août je vous propose
               trois hôtels le Sofitel palais des congrès hôtel de la Gardanne Mercure
               Saint Charles souhaitez vous réserver dans l' un de ces hôtels ou
               obtenir des informations supplémentaires}[]

365_12_spk @command-tache{alors j' ai des informations supplémentaires}[information]
           @null{je voudrais}[null] @nombre-chambre{une chambre}[1]
           @sejour-nbCouple{pour un couple}[1] @sejour-nbAdulte{deux adultes}[2]
           @comparatif-paiement{moins de}[inferieur]
           @paiement-montant-entier{deux cents}[200] @paiement-monnaie{euros}[euro]

365_13_woz @woz{merci de patienter je recherche cette information à l' hôtel Sofitel
               palais des congrès la chambre double est à deux cent cinquante euros à
               l' hôtel de la Gardanne la chambre double est à cent vingt cinq euros
               au Mercure Saint Charles la chambre double est à soixante quinze euros
               souhaitez vous réserver dans l' un de ces hôtels ou obtenir d' autres
               informations}[]

365_14_spk @command-tache{réserver}[reservation] @null{dans}[null]
           @nombre-hotel{l' un}[1] @lienRef-coRef{de ces}[pluriel]
           @objetBD{hôtels}[hotel]

365_15_woz @woz{vous souhaitez réserver dans quel hôtel}[]

365_16_spk @nom{Gardanne}[gardanne]

365_17_woz @woz{merci de patienter je vérifie les disponibilités à l' hôtel Gardanne
               une chambre est disponible correspondant à vos critères la chambre
               est à cent vingt cinq euros est ce que je vous réserve cette
               chambre}[]
```

Figura 7.5: *El mismo fragmento de la figura 7.4 de una interacción usuario-máquina del corpus French Media. Se indican los conceptos con su soporte y su valor usando esta nomenclatura: concepto*{ *soporte* }[*valor*]*. Al comienzo de cada frase se indica el número del diálogo, el turno y si se trata del usuario (spk) o de la máquina (woz). //* Same fragment with *concept*{ *support* }[*value*] indications. Referenced at page 306.

Tabla 7.1: *Parámetros de los modelos acústicos HMM/ANN para el corpus Media. //* HMM/ANN acoustic models parameters. Referenced at page 306.

Replacement:	$300K$ training
	$200K$ validation
# of phonemes:	35
# states per phoneme:	3
Acoustic frame size:	12 components
MLP context:	8 frames left
	current frame
	8 frames right
MLP topology:	$(8+1+8) \times 12 - 400 - 400 - 3 \times 35$
MLP initial weights range:	$[-0.4, 0.4]$
Learning Rate:	0.002
Momentum:	0.005
Weight Decay:	1×10^{-10}
Gaussian noise:	$(\mu = 0, \sigma = 0.02)$

modelos híbridos HMM/ANNs [Bourlard & Morgan, 1994] con topología de 3 estados y sin "skips".

La red neuronal de los modelos acústicos HMM/ANN es un perceptrón multicapa (MLP) que recibe a su entrada 17 vectores de características, el vector actual, y 8 a cada lado del mismo. Cada vector está formado por 12 características, con lo que la entrada de la red está formada por 204 neuronas. La red neuronal tiene dos capas ocultas de 400 neuronas cada una y con función de activación sigmoide. A la salida hay 105 neuronas (3 estados por cada fonema), y la función de activación es la softmax. Se ha utilizado el algoritmo de entrenamiento con reemplazamiento que permite que en cada época la red neuronal visite un 50 % de datos del Media y un 50 % de datos de Ester-II. Los patrones de la entrada de la red neuronal son modificados en cada iteración del algoritmo de aprendizaje añadiendo un pequeño ruido Gaussiano para aumentar la capacidad de generalización y evitar el sobreentrenamiento.

Los modelos acústicos HMM/ANN han sido entrenados siguiendo el algoritmo EM basado en Viterbi del capítulo 6. La tabla 7.1 muestra los parámetros del algoritmo de entrenamiento y de la red utilizada.

7.5 Comprensión del lenguaje en sistemas de diálogo

El proceso de comprensión del lenguaje consiste en convertir una secuencia de palabras en una secuencia conceptos que facilite el procesamiento automático de la señal hablada. Para ello, definimos un vocabulario V_C de *etiquetas de conceptos* $\{C_1, \ldots, C_{|V_C|}\}$, de manera que cada etiqueta representa un determinado conocimiento en el dominio de la aplicación. Llamaremos *soporte* a la secuencia $\bar{\sigma}_m$ de palabras que se asocia a un determinado concepto C_m en una determinada frase. De esta forma podemos asociar a una secuencia de palabras su correspondiente secuencia de conceptos (asociando cada concepto con su soporte) que nos permitiría describir el conocimiento de dicha secuencia. Como ejemplo, la figura 7.6 ilustra un fragmento de una frase de la tarea Media, y su correspondiente secuencia de conceptos, asociando a cada concepto su soporte formado por un segmento de la frase.

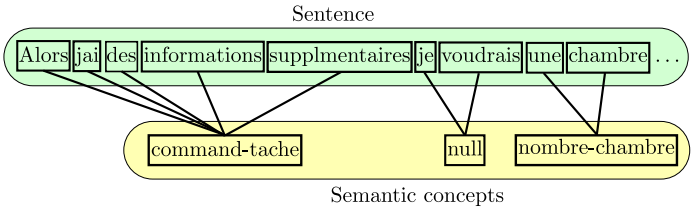


Figura 7.6: *Ejemplo de asociación entre palabras de una frase y su correspondiente etiquetado semántico en la tarea Media. //* Example of words and semantic labels for the Media task. Referenced at page 307.

7.5.1. Comprensión del lenguaje mediante LMs

Es posible transformar el conjunto de secuencias de conceptos con su soporte, en una secuencia de pares *(concepto, palabra)*,

$$(c, w)_{u,1}, \ldots, (c, w)_{u,k}, \ldots, (c, w)_{u,N(u)} \,,$$

correspondiente al turno de diálogo u formado por $N(u)$ palabras.[2] Conocida esta secuencia de pares, es posible establecer un nuevo vocabulario de instancias de estos pares, y con él aprender un modelo de lenguaje sobre la unidad del par *(concepto, palabra)*.

El conjunto de pares *(concepto, palabra)*, (c, w), que aparecen en la partición de entrenamiento del corpus Media es una pequeña porción del conjunto de todos los posibles pares de $\Omega_C \times \Omega$, y consta de $\Omega_t = 4\,207$ pares.

Una vez entrenado el modelo de lenguaje, es posible lanzar un proceso de reconocimiento que, a partir de la señal acústica y de forma integrada, produzca la secuencia de pares de mayor probabilidad. Este modelo de lenguaje computa la probabilidad conjunta de ambas secuencias, y la búsqueda implementada en la decodificación obtiene aquella secuencia de pares que maximiza la probabilidad de ambas fuentes de conocimiento de forma conjunta. Formalmente, la ecuación fundamental (1.2) puede escribirse para esta tarea de la siguiente forma:

$$(\hat{\bar{c}}, \hat{\bar{y}}) = \underset{(\bar{c},\bar{y})\in\Omega_t^+}{\arg\,\max}\, p(\bar{c}, \bar{y}|\bar{x}) = \tag{7.1}$$

$$= \underset{(\bar{c},\bar{y})\in\Omega_t^+}{\arg\,\max}\, p(\bar{x}|\bar{c}, \bar{y}) p(\bar{c}, \bar{y}) \,, \tag{7.2}$$

donde $\bar{c} = c_1 c_2 \ldots c_{|\bar{y}|}$ es la secuencia de conceptos, uno por cada palabra de $\bar{y}$ y Ω_t es el vocabulario de pares *(concepto, palabra)*.

7.5.2. Cache NN LMs para tareas de diálogo hablado

El objetivo de este capítulo es observar el efecto que tienen los modelos Cache NN LMs, presentados en la sección 5.1, en una tarea de diálogo hablado. En esta tarea utilizaremos Cache NN LMs que aprovecharán la historia del diálogo actual para introducirla en la caché del

[2] En la sección 5.1 un diálogo es una interacción completa con el usuario, y un turno de diálogo es una frase dentro de dicha interacción.

modelo de lenguaje, permitiendo que el modelo se adapte a la situación y contexto del diálogo. Siguiendo la ecuación 5.3, un Cache NN LM computa la probabilidad de una secuencia de pares *(concepto, palabra)*:

$$p(\bar{y}) \approx p(\bar{y}|\bar{h}_1^{u-1}) \approx \prod_{j=1}^{|\bar{y}|} p(y_j|\bar{h}_j, \bar{h}_1^{u-1}) \,, \tag{5.3}$$

siendo $\bar{h}_1^{u-1}$ el módulo caché del modelo, tal que, siendo u el índice de la interacción actual, resume todas las interacciones con el usuario desde la 1 hasta la $u-1$.

Representación de la historia del diálogo en Cache NN LMs

Se establecen cuatro formas diferentes de representar la historia del diálogo, diferenciando cada una de ellas según las fuentes de información utilizadas para generar la caché:

A) Sólo conceptos: la caché únicamente contiene conceptos semánticos de la historia del diálogo. Estos se extraen a partir de las hipótesis reconocidas por el sistema, extrayendo los conceptos de los pares *(concepto, palabra)* y desechando las palabras.

B) Sólo palabras: en este caso limitamos la caché a contener una representación de la historia del diálogo basada en las palabras reconocidas por el sistema. Estas palabras se extraen de las secuencias de pares *(concepto, palabra)* que genera el sistema de SLU, desechando las palabras.

C) Conceptos y palabras: tomaremos en consideración para la caché tanto palabras como conceptos, pero cada uno de ellos por separado. Puede entenderse como la concatenación de cada una de las cachés obtenidas por las dos configuraciones anteriores.

D) Conceptos y palabras, y además las palabras del WOZ: idéntico a la configuración anterior, pero tomando en consideración para alimentar la caché también las respuestas generadas por el sistema de diálogo. De esa forma no solo tenemos en cuenta las palabras del usuario, sino, también podemos tener en cuenta las respuestas del sistema.

7.5.3. Arquitectura de los Cache NN LMs

Un Cache NN LM es un NN LM que estima la probabilidad del par *(concepto, palabra)* del turno u-ésimo usando los $(N-1)$ pares que le preceden, con una entrada adicional que denominamos caché y que modela la historia del diálogo en los turnos anteriores al u-ésimo, tal y como ha sido descrito en la sección 5.1. Nótese que para los casos descritos en la sección anterior, los Cache NN LMs tendrán la misma salida, y sólo difieren en el tamaño y contenido de la entrada que hace de módulo caché.

Los pares *(concepto, palabra)* que aparecen una única vez en el conjunto de entrenamiento se han eliminado del vocabulario utilizado para los Cache NN LMs, usando todo ese conjunto de palabras para entrenar la neurona OOS de entrada y la neurona OOS de salida. El vocabulario queda reducido a $\Omega_t' = 2\,559$ pares *(concepto, palabra)* diferentes. No obstante, el vocabulario completo de $\Omega_t = 4\,207$ pares *(concepto, palabra)* ha sido utilizado para entrenar modelos de lenguaje estándar usando el descuento de Kneser-Ney modificado, con lo que la cobertura del vocabulario completo de pares está garantizada.

7.6 Sistema de comprensión automática del habla

El sistema completo de SLU que proponemos en este trabajo está compuesto de varias etapas, como se ilustra en la figura 7.7:

1. **Etapa de ASR**: el módulo encargado de reconocimiento del habla recibe una señal acústica, y realiza la decodificación siguiendo el algoritmo de dos pasos descrito en el capítulo 6 y un modelo de lenguaje de pares *(concepto, palabra)*. El léxico utilizado para la decodificación está compuesto por los pares *(concepto, palabra)* del conjunto de entrenamiento del corpus Media. La salida de esta etapa es la frase formada por pares *(concepto, palabra)* cuya probabilidad conjunta es la máxima, calculada por el algoritmo de búsqueda, basado en Viterbi, descrito en la sección 6.2.

   En el ejemplo de la figura 7.7 la salida de esta etapa se corresponde con la siguiente secuencia de pares *(concepto, palabra)*:

   ```
   null/oui
   temps-range/la
   temps-range/troisieme
   temps-unite/semaine
   temps-mois/d'
   temps-mois/aout+
   ```

2. **Etapa de etiquetado BIO**: utilizamos un modelo estadístico "Conditional Random Field" (CRF) [Lafferty *et al.*, 2001] muy simple para etiquetar cada par *(concepto, palabra)* como:

   - Begin (B): indica que la palabra del par es comienzo del soporte del concepto;
   - Inside (I): indica que la palabra del par pertenece al soporte del concepto que le acompaña en el par, pero no es la primera.
   - Out (O): indica que dicho par no es de interés de cara a la etapa de comprensión.

   Este etiquetado permitirá obtener la secuencia final de conceptos. El CRF ha sido estimado mediante la herramienta CRF++[3].

   Siguiendo con el ejemplo de la figura, la salida de esta etapa es:

   ```
   null/oui              O
   temps-range/la        B
   temps-range/troisieme I
   temps-unite/semaine   B
   temps-mois/d'         B
   temps-mois/aout       I
   ```

3. **Etapa de "chunking"**: dada la secuencia de pares etiquetados con B, I, O un sencillo algoritmo agrupa aquellas subsecuencias de pares que son susceptibles de formar un par concepto con su soporte, siendo el soporte una secuencia de una o más palabras. La salida de esta última etapa proporciona tanto la secuencia de conceptos que han sido reconocidos por el sistema, como la asociación a cada concepto de su secuencia de palabras que le hace de soporte.

[3] http://crfpp.sourceforge.net

Tabla 7.2: *PPL para los conjuntos de validación y test del corpus Media de los modelos de lenguaje utilizados para validar el sistema de ASR. //* PPL of the validation and test sets for the corpus Media. Referenced at page 307.

	2-grams		*3-grams*		*4-grams*	
LM	*Val.*	*Test*	*Val.*	*Test*	*Val.*	*Test*
Word-based count-based N-gram	29.6	27.4	23.3	21.1	22.7	20.6
Word-based NN LM	22.2	20.2	20.2	18.5	**19.6**	**18.1**

El resultado de esta etapa, continuando con el ejemplo de la figura, es:

```
temps-range{la troisieme} temps-unite{semaine} temps-mois{d' aout}
```

7.7 Experimentación

La experimentación con modelos tipo caché para tareas de SLU va a dividirse en dos apartados. Primero presentaremos los resultados de los modelos acústicos descritos en la sección 7.4, lo que nos dará una referencia de la calidad de nuestro sistema de ASR. En segundo lugar presentaremos resultados a nivel de Concept Error Rate (CER) al utilizar modelos tipo caché durante la etapa de decodificación, y usando la aproximación a SLU basada en modelos de lenguaje de pares *(concepto, palabra)*.

7.7.1. Resultados del sistema baseline de ASR

Con la intención de establecer la calidad del sistema de ASR desarrollado, se han estimados varios modelos de lenguaje, todos ellos entrenados con la partición de entrenamiento del corpus Media, compuesta por un vocabulario de 2 007 palabras. Se han probado combinaciones de $N = 2, 3, 4$ para el orden del N-grama, y se han comparado modelos entrenados con la herramienta SRI [Stolcke, 2002] usando el descuento de Kneser-Ney modificado, y NN LMs configurados para los mismos valores de N. Cada NN LM es la combinación de cuatro redes neuronales (capa de proyección con 128, 160, 192 y 224 neuronas, capa oculta con 200 neuronas) y el modelo de 4-gramas estándar entrenado con SRI. La tabla 7.2 muestra los resultados en PPL, y la tabla 7.3 los resultados en WER obtenidos para las particiones de validación y test del Media. Se puede observar en la misma que utilizar NN LMs permite mejorar en 1.2 puntos absolutos de WER el baseline que utiliza modelos de lenguaje estándar. La mejora es de un $3.8\,\%$ en términos relativos.

A efectos comparativos, en [Hahn *et al.*, 2011] el WER de su sistema de ASR es de $30.3\,\%$ en el conjunto de validación y $31.4\,\%$ en el conjunto de test. La comparación de estos resultados con el baseline aquí presentado permite afirmar que este sistema de ASR es *competitivo* con el estado del arte.

7.7.2. Resultados de comprensión automática del habla

Se han realizado varias series de experimentos utilizando los modelos tipo caché descritos en el apartado 7.5.3, y que hemos denotado así:

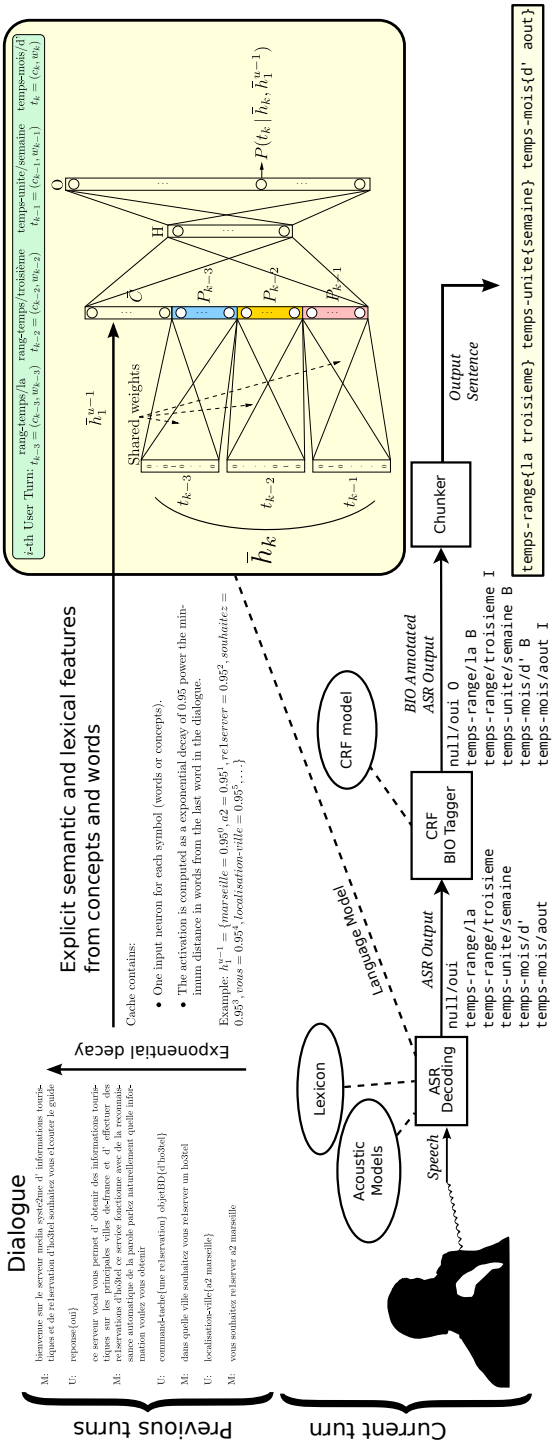


Figura 7.7: *Esquema del sistema completo de comprensión automática del habla para la tarea de diálogo del corpus Media.* // Scheme of the SLU system for the Media task. Referenced at page 307.

Tabla 7.3: *Resultados en % WER para los conjuntos de validación y test del corpus Media con los modelos de lenguaje utilizados para validar el sistema de ASR. //* WER of the validation and test sets for the corpus Media. Referenced at page 307.

	2-grams		*3-grams*		*4-grams*	
LM	*Val.*	*Test*	*Val.*	*Test*	*Val.*	*Test*
Word-based count-based N-gram	35.2	34.7	31.8	31.4	31.5	31.3
Word-based NN LM	31.6	31.1	30.7	30.6	**30.4**	**30.1**

1. Dos modelos de lenguaje sin caché, es decir, que estima $p(y_j|\bar{h}_j)$ donde $\bar{h}_j$ está formado por los $N-1$ pares *(concepto, palabra)* anteriores. Se han estimado para obtener resultados de referencia:
   - baseline-a: modelo de lenguaje de pares *(concepto, palabra)* estimado utilizando el descuento de Kneser-Ney modificado mediante SRI.
   - baseline-b: modelo de lenguaje de pares *(concepto, palabra)* estimado mediante un NN LM, combinado linealmente con el anterior.
2. Cuatro modelos Cache NN LM para estimar $p(y_j|\bar{h}_j, \bar{h}_1^{u-1})$, que se diferencian en la información representada en la historia $\bar{h}_1^{u-1}$, según se ha descrito en la sección 7.5.2:
   - cacheNNLM-A: modelo de lenguaje de pares *(concepto, palabra)* estimado mediante un Cache NN LM, combinado linealmente con baseline-a, utilizando sólo información relativa a los conceptos en la caché.
   - cacheNNLM-B: modelo de lenguaje de pares *(concepto, palabra)* estimado mediante un Cache NN LM, combinado linealmente con baseline-a, utilizando sólo información relativa a las palabras en la caché.
   - cacheNNLM-C: modelo de lenguaje de pares *(concepto, palabra)* estimado mediante un Cache NN LM, combinado linealmente con baseline-a, utilizando información relativa a los conceptos y a las palabras en la caché.
   - cacheNNLM-D: modelo de lenguaje de pares *(concepto, palabra)* estimado mediante un Cache NN LM, combinado linealmente con baseline-a, utilizando en la caché información relativa a los conceptos, a las palabras y también las palabras que el sistema ha producido como respuesta al usuario.

Las redes neuronales han sido entrenadas usando el algoritmo BP descrito en el capítulo 2, utilizando el término de regularización weight decay y la función de error entropía cruzada. Cada NN LM o Cache NN LM es una combinación de cuatro redes neuronales que difieren en el tamaño de la capa de proyección por palabra (128, 160, 192, 224). La capa oculta está formada por 200 neuronas.

Utilizando estos modelos se ha calculado por un lado el WER del sistema, que se muestra en la tabla 7.4, y por otro lado el CER[4], en la tabla 7.5. Para obtener el WER basta con

[4]En este capítulo CER hace referencia a Concept Error Rate, que no debe ser confundido con Character Error Rate.

Tabla 7.4: *Resultados en % WER para los conjuntos de validación y test del corpus Media usando los modelos de lenguaje extendidos a pares* (concepto, palabra) *usados para SLU, con y sin caché. //* WER of the validation and test sets of Media, using LMs of (concept,word) pairs. Referenced at page 307.

	2-grams		*3-grams*		*4-grams*	
System	*Val.*	*Test*	*Val.*	*Test*	*Val.*	*Test*
baseline-a	34.1	33.0	32.8	32.4	32.3	32.1
baseline-b	33.0	31.8	31.8	31.1	31.4	32.2
cacheNNLM-A	31.8	31.6	30.6	31.2	30.5	30.9
cacheNNLM-B	31.7	32.0	31.2	30.9	30.5	**30.5**
cacheNNLM-C	31.9	31.8	31.8	31.4	31.2	31.2
cacheNNLM-D	31.2	31.4	31.4	30.7	30.5	**30.5**

eliminar de la salida del sistema la etiqueta de concepto y quedarnos con las palabras soporte de cada uno[5]. Los resultados obtenidos indican que utilizar una caché permite no solo mejorar los resultados en términos de comprensión, sino que también mejora los resultados en reconocimiento, el WER se reduce hasta 30.5 puntos en Test, frente a los 30.1 de WER del sistema basado en un NN LM de palabras (véase la tabla 7.3).

El Concept Error Rate (CER) se ha obtenido eliminando aquellos conceptos que han sido etiquetados como O, y eliminando los soportes, es decir, medimos el CER a nivel de etiquetas conceptuales, sin importarnos el valor asociado a dicha etiqueta[6].

Los resultados obtenidos por los Cache NN LMs *mejoran*, de forma estadísticamente significativa, el baseline, ya que el intervalo de confianza al $95\,\%$ en el conjunto de test es de 1.3 puntos absolutos de CER. En el mejor de los casos se ha disminuido en 3.2 puntos absolutos de CER el modelo baseline-a, y en 2.0 puntos absolutos de CER el baseline-b. Esto es, una mejora relativa del $10.9\,\%$ y del $7.1\,\%$ respectivamente.

Estos resultados se acercan mucho a los obtenidos en [Hahn *et al.*, 2011], pero utilizando un sistema de ASR mucho más simple y un sistema de comprensión también más sencillo. La tabla 7.6 muestra los resultados presentados en [Hahn *et al.*, 2011] sobre la misma tarea y corpus. Se puede ver que el mejor resultado es de $23.8\,\%$ de CER usando CRFs. En esta tabla de resultados el modelo aquí presentado se situaría en un tercer puesto, equivalente a nivel de significancia estadística con los sistemas basados en SVMs y MEMMs. No obstante, los resultados están todavía lejos del CER obtenido por los CRFs.

7.8 Conclusiones y trabajo futuro

Los resultados en comprensión han mostrado una mejora robusta en CER, estadísticamente significativa si comparamos los modelos conexionistas con las técnicas estándar en modelado de lenguaje.

Estos resultados utilizando modelos conexionistas en una arquitectura de ASR relativamente sencilla, sugieren la extensión del modelo conexionista a arquitecturas más complica-

[5] Siguiendo el ejemplo de la figura 7.7, se mediría el WER de la secuencia reconocida "la troiesème semaine d' août".

[6] Volviendo al ejemplo de la figura 7.7, se mediría el CER de la secuencia de conceptos "temps-range temps-unite temps-mois".

Tabla 7.5: *Resultados en % CER para los conjuntos de validación y test del corpus Media usando los modelos de lenguaje extendidos a pares* (concepto, palabra) *usados para SLU, con y sin caché. //* CER of the validation and test sets of Media using LMs of (concept,word) pairs. Referenced at page 307.

	2-grams		*3-grams*		*4-grams*	
System	*Val.*	*Test*	*Val.*	*Test*	*Val.*	*Test*
baseline-a	33.6	30.1	32.9	29.3	33.5	29.3
baseline-b	33.1	28.3	30.7	27.4	30.2	28.1
cacheNNLM-A	31.7	28.2	29.7	27.0	29.7	27.0
cacheNNLM-B	30.5	27.3	29.7	27.0	30.0	**26.1**
cacheNNLM-C	32.2	28.3	30.5	27.0	30.8	27.4
cacheNNLM-D	31.2	28.2	29.9	26.2	30.3	27.1

Tabla 7.6: *Resultados en % CER con el corpus Media presentados en [Hahn* et al.*, 2011] para SLU. //* CER presented in [Hahn *et al.*, 2011] for the Media task. Referenced at page 307.

System	*Val.*	*Test*
CRF	24.0	23.8
SVM	27.1	25.8
MEMM	26.6	26.4
FST	28.3	27.5
SMT	28.4	29.0
DBN	29.5	29.1

das, utilizando esquemas de SLU donde sea posible hacer la decodificación en conceptos a partir de grafos de palabras o redes de confusión ("Confusion Networks"). Se ha calculado un oráculo con la salida del sistema para tener una cota optimista del CER. Ha consistido en calcular el porcentaje de conceptos que aparecen en la referencia y no en la salida del sistema, sin tener en cuenta el orden de los mismos. El oráculo ha dado un error del $15.9\,\%$ para el mejor baseline de N-gramas, y de $14.2\,\%$ para la mejor de las Cache NN LM, sugiriendo la existencia un gran margen de mejora.

7.9 Resumen

En este capítulo se ha descrito la aplicación de los Cache NN LMs a tareas de comprensión del lenguaje, donde el uso de la caché por parte del modelo de lenguaje permite que adapte su distribución de probabilidad haciéndola dependiente del contexto pasado mucho más allá de las dos o tres palabras anteriores. El capítulo ha cumplido con estos objetivos tecnológicos:

- Desarrollar un sistema de ASR que permita hacer investigaciones en este área. El sistema ha demostrado ser competitivo con el estado del arte en ASR.
- Extender el modelado de lenguaje de una forma simple y sencilla para poder llevar a cabo tareas de comprensión automática del habla utilizando los sistemas y modelos desarrollados. Se ha mejorado entre un $7.1\,\%$ y $10.9\,\%$ los sistemas baseline estimados para esta tarea.

El objetivo científico más importante ha sido utilizar los Cache NN LMs en una aplicación práctica, en este caso, comprensión en un sistema de diálogo hablado. Los resultados obtenidos sugieren la necesidad de seguir trabajando en la línea utilizando esquemas de SLU y de ASR más complejos. Este trabajo ha sido publicado en el ICASSP del 2012 [Zamora-Martínez *et al.*, 2012a].

8

RECONOCIMIENTO DE ESCRITURA MANUSCRITA OFF-LINE

Índice

Este capítulo detalla la integración de NN LMs en un sistema automático de reconocimiento de escritura manuscrita (HTR) basado en modelos de Markov hibridados con redes neuronales. El procedimiento de decodificación utiliza los módulos `dataflow` correspondientes al reconocimiento de secuencias del capítulo 6. El HTR es una tarea de gran relevancia en la actualidad. La mejora de los resultados en este campo tiene una relación directa con el esfuerzo necesario para la transcripción de las bases documentales en papel que a día de hoy están siendo transcritas en el marco de diversos proyectos de investigación. Este capítulo estudia el impacto del uso de NN LMs dentro de este campo. De hecho, hemos sido los primeros en usar NN LMs en tareas de HTR [Zamora-Martínez *et al.*, 2010; España-Boquera *et al.*, 2011; Frinken *et al.*, 2012; Zamora-Martínez *et al.*, 2012c].

En tareas de HTR es posible utilizar modelos de grafemas (o letras), en lugar de palabras, logrando sistemas de reconocimiento donde el léxico no está definido y se forma mediante la concatenación de grafemas. También en este caso la aplicación de NN LMs ofrece una ventaja clara frente a los modelos de lenguaje de N-gramas estándar.

Los objetivos de este capítulo son:

- Introducción al HTR.
- Obtención de un sistema baseline estado del arte dentro del campo.
- Mejora de dicho sistema baseline mediante la integración de NN LMs.
- Comparación de la aproximación integrada de NN LMs, utilizando la técnica vista en el capítulo 4, frente al rescoring de listas de N-best, evaluando tanto la calidad del sistema como la eficiencia del mismo.

Adicionalmente se presentan resultados preliminares de un modelo de lenguaje basado en redes neuronales recurrentes. Dichos resultados son parte del trabajo que se ha realizado en colaboración con el IAM de la Universidad de Bern.

8.1 Introducción

El reconocimiento de escritura es hoy en día uno de los campos más importantes en reconocimiento de formas. El reconocimiento de escritura se puede clasificar en dos grandes áreas, que dependen de la forma en que los datos son capturados:

- **On-line**: en este caso los datos de entrada se capturan mediante un lápiz de tinta electrónica, o bien una pizarra electrónica, o cualquier otro dispositivo que permita registrar el movimiento del trazo conforme avanza en el tiempo la escritura. En este caso los sistemas de reconocimiento están obteniendo muy buenos resultados, entre otras razones debido a que la información temporal del trazo permite distinguir mejor entre unas letras y otras.
- **Off-line**: se conoce con este nombre al reconocimiento de escritura cuando el origen de los datos es una imagen del texto. En este caso la información temporal de los trazos se ha perdido, lo que aumenta la complejidad y el error final del sistema.

El trabajo que aquí se presenta consiste en aplicar modelos conexionistas de lenguaje al reconocimiento de escritura off-line. La aproximación que seguiremos en este capítulo está basada en modelos ocultos de Markov hibridados con redes neuronales (HMM/ANN). Por lo tanto el sistema final es un híbrido de técnicas estadísticas y conexionistas en todas las etapas del reconocimiento, ya que está compuesto por un sistema HMM/ANN para los modelos ópticos, y por un modelo conexionista de lenguaje combinado con uno estadístico estándar.

Al igual que en el capítulo anterior, la formalización parte de la ecuación fundamental (1.2):

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{y}|\bar{x}) = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y}) \tag{1.2}$$

donde tras aplicar la regla de Bayes aparece el término $p(\bar{x}|\bar{y})$ que en este caso se denomina modelo óptico, donde $\bar{x}$ es la secuencia de vectores de características extraídos a partir de imágenes de texto, e $\bar{y}$ es la secuencia de palabras que estamos evaluando como hipótesis. $p(\bar{y})$ es la probabilidad a priori estimada por el modelo de lenguaje para la hipótesis $\bar{y}$. De nuevo se puede interpretar esta ecuación dentro del marco de la máxima entropía, en cuyo caso tendríamos una combinación de tres modelos: el modelo óptico, el modelo de lenguaje, y el número de palabras de la hipótesis. Por lo tanto, igual que en el capítulo anterior, a través de la ecuación (1.3) aparecen los parámetros GSF (λ_1) y WIP (λ_2) que modulan la combinación de los modelos. La ecuación (1.4) sigue esta aproximación, y la retomamos aquí:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y})^{\lambda_1} exp(|\bar{y}|)^{\lambda_2} \,. \tag{1.4}$$

Un trabajo de interés sería estudiar cómo afecta la combinación de modelos ópticos diferentes de esta manera, y al mismo tiempo con modelos de lenguaje diferentes, entre ellos modelos de categorías, etiquetas POS, etc.

8.2 Preproceso de escritura manuscrita

La escritura manuscrita precisa de un preproceso sistemático que permita reducir la variabilidad intraclase, aumentando la variabilidad interclase cuando sea posible. Para lograr

Figura 8.1: *Imagen de una línea de texto con las áreas de referencia (ascendentes, descendentes, cuerpo central) y las líneas de referencia (líneas base superior e inferior, línea de ascendentes y línea de descendentes) [España-Boquera* et al.*, 2011]. //* Text line image with its reference areas and reference lines.

este objetivo se sigue el procedimiento que describimos a continuación, diferenciado en dos grandes bloques: a nivel de documento y a nivel de línea.

- Preproceso a nivel de documento:
  1. limpieza de las imágenes: se eliminan ruidos y manchas;
  2. corrección del "skew": se elimina la rotación que pudiera haber sufrido el documento al ser escaneado;
  3. extracción de líneas: se detectan las regiones del documento escaneado que se corresponden con líneas;
  4. otros preprocesos a nivel de documento: en determinadas tareas puede interesar detectar figuras, columnas de texto, la dirección del flujo de lectura del documento, ...
- Preproceso a nivel de línea:
  1. corrección del "slope": esta fase elimina la rotación que pudieran tener las palabras, ajustando a la horizontal todas las líneas base inferior de las palabras (véanse las figuras 8.2(d) y 8.2(e));
  2. corrección del "slant": se elimina la inclinación de las letras, ajustándola al eje vertical (véanse las figuras 8.2(e) y 8.2(f));
  3. normalización del tamaño: se normaliza la proporción de las letras para que sea homogénea (véanse las figuras 8.2(e) y 8.2(h)).

Una vez realizado todo este procedimiento se pasa a ejecutar el proceso de extracción de características. En este trabajo asumimos que el preproceso del documento ya ha sido realizado y por lo tanto disponemos directamente de imágenes que son líneas de texto escaneado. El preproceso de las líneas y posterior entrenamiento de los HMM/ANN aquí presentados se describe en [Gorbe *et al.*, 2008; España-Boquera *et al.*, 2011], si bien en este trabajo se han mejorado algunos aspectos técnicos que permiten obtener pequeñas mejoras en las prestaciones finales.

8.2.1. Detección de puntos de interés

La mayoría de los sistemas de reconocimiento requieren la detección de las diferentes áreas en las que podemos dividir la escritura manuscrita: cuerpo central (el área que queda entre la línea base superior y la inferior), los ascendentes, y los descendentes (véase la

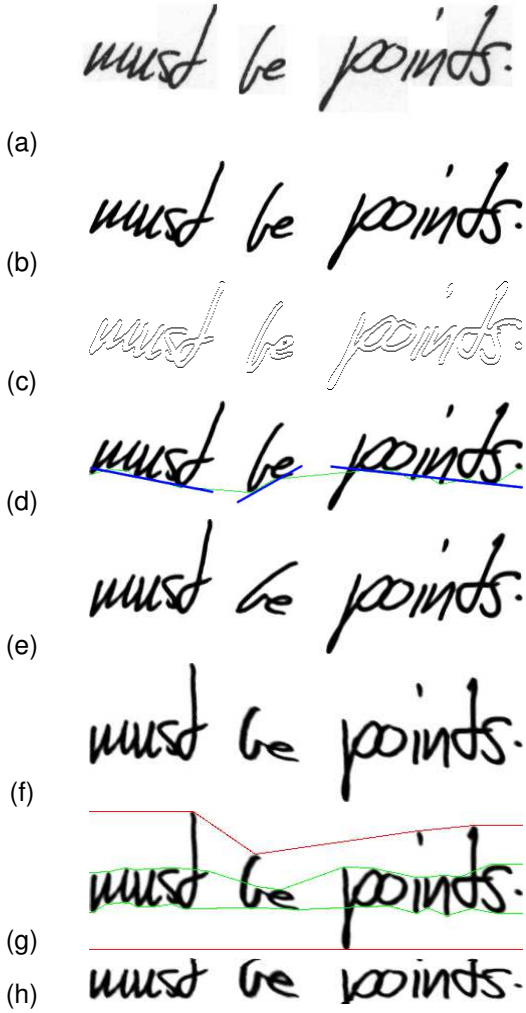


Figura 8.2: *Ejemplo de preproceso [España-Boquera* et al.*, 2011]. De arriba a abajo: (a) Imagen original de la IAM-DB. (b) Imagen limpiada. (c) Contornos del texto para extraer los extremos locales. (d) Extremos locales clasificados como línea base inferior mediante un MLP, usados para corregir el slope. (e) Imagen con el slope corregido. (f) Imagen con el slant corregido. (g) Extremos locales etiquetados por un MLP para extraer las 4 líneas de referencia usadas para la normalización del tamaño. (h) Imagen normalizada en tamaño. //* Preprocess example: (a) Original image. (b) Cleaned image. (c) Text contours. (d) Local extrema classified as lower baseline. (e) Slope corrected image. (f) Slant corrected image. (g) Local extrema classified as reference lines. (h) Text size normalized image.

figura 8.1). Estas áreas se pueden detectar mediante histogramas de la proyección horizontal [Burr, 1982; Bozinovic & Srihari, 1989; Vinciarelli & Luettin, 2001] o bien obteniendo los contornos superior e inferior de la imagen [Wong *et al.*, 1982; Romero *et al.*, 2006]. En este trabajo se sigue una aproximación basada en la detección de puntos de la imagen, y su posterior clasificación mediante técnicas de aprendizaje automático supervisado [Simard *et al.*, 2005; Gorbe *et al.*, 2008]. Estos puntos son los extremos verticales locales de cada contorno del texto, y sirven para determinar las líneas de referencia: línea de ascendentes, línea base superior, línea base inferior, y línea de descendentes (véase figura 8.2). Estas líneas permiten eliminar el slope y realizar la normalización del tamaño de una forma eficiente.

Tras limpiar las imágenes para eliminar ruidos molestos, mediante técnicas basadas en redes neuronales [Hidalgo *et al.*, 2005; Zamora-Martínez *et al.*, 2007], se extraen los extremos locales. Primero se obtiene el contorno vertical de la imagen (véase figura 8.2(c)). Tras esto, los puntos del contorno se agrupan en líneas siguiendo un criterio de vecindad: dos píxeles en columnas adyacentes se consideran de la misma línea cuando la diferencia en la coordenada vertical de ambos es menor que un determinado umbral (en nuestro caso, 3 píxeles). Finalmente, se extraen los máximos locales del contorno superior, y los mínimos locales del contorno inferior.

8.2.2. Corrección del slope

Para corregir el slope basta con conocer la línea base inferior. Con este fin se entrena un perceptrón multicapa (MLP) para clasificar los extremos locales como pertenecientes o no a la línea base inferior. Denotaremos con Slope-MLP a este perceptrón multicapa. La entrada de Slope-MLP es una ventana deslizante centrada en el píxel a clasificar.

Conocidos los puntos de la línea base inferior, la imagen se divide horizontalmente en bloques. Se aplica la corrección del slope a cada bloque, tomando los puntos clasificados como línea base inferior en dicho bloque para ajustar por mínimos cuadrados una recta. La inclinación de dicha recta nos indica la rotación que debemos aplicar a la imagen para que la inclinación final de la recta sea de 0^o. Un ejemplo del proceso se puede ver en la figura 8.2(d). La figura 8.2(e) muestra un ejemplo de una imagen con el slope corregido.

8.2.3. Corrección del slant

Tras corregir el slope, es necesario eliminar el slant. Esta corrección se lleva a cabo en dos pasos:

1. Se corrige la imagen estimando el ángulo de slant global. Para ello se aplica una transformación "shear" a la imagen para cada valor entero del ángulo en el intervalo [-50^o,50^o], eligiendo aquel ángulo que hace máxima la varianza de la proyección vertical [Pastor *et al.*, 2004].

2. Se aplica una corrección no uniforme del slant basada en técnicas de aprendizaje automático supervisado. Se utiliza un MLP, que llamaremos Slant-MLP, entrenado para estimar si una determinada columna de texto de la imagen tiene o no slant. Se usa como entrada una ventana deslizante, y para cada columna se aplica una corrección del slant mediante shear, tomando ángulos enteros, de nuevo en el intervalo [-50^o,50^o], y se apunta el valor de la salida la red neuronal. Esta salida representa la confianza que tiene el Slant-MLP de que la columna correspondiente de píxeles tenga o no el ángulo

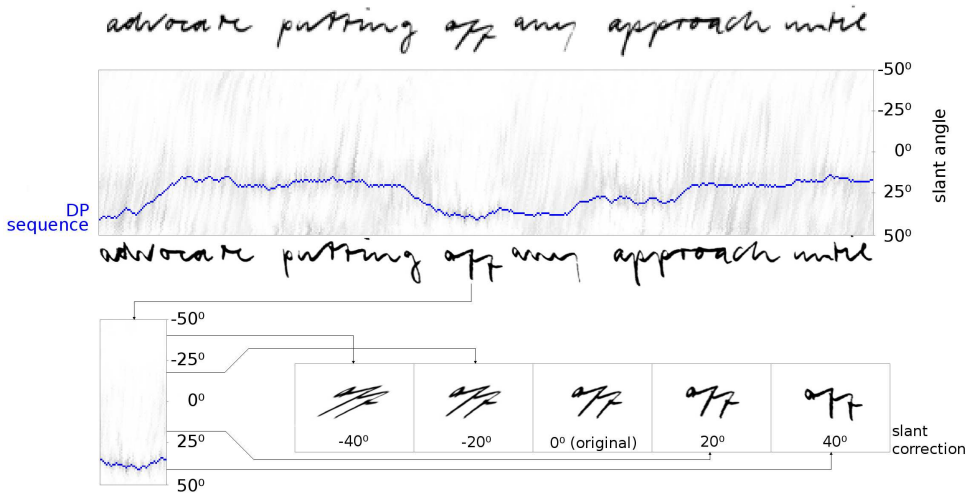


Figura 8.3: *Ejemplo del procedimiento de corrección del slant [España-Boquera* et al.*, 2011]: imagen de texto original extraído de IAM-DB (arriba) y su correspondiente imagen de texto sin slant (abajo). El Slant-MLP estima una medida de bondad del ángulo de slant para cada columna de píxeles, que se ilustra mediante una matriz de niveles de gris. Un algoritmo de programación dinámica obtiene la secuencia óptima de ángulos de slant. //* Slant correction example.

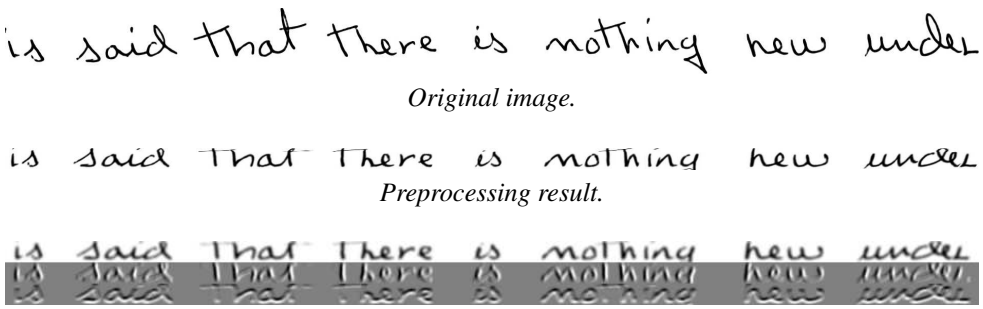
de slant dado. Con esta información, por cada posible columna de la imagen, y por cada posible ángulo, podemos crear una matriz de tamaño *Ancho de la imagen*$\times 101$. Mediante un sencillo algoritmo de programación dinámica [Uchida *et al.*, 2001] se puede obtener un camino que haga máxima la suma de la confianza en cada columna, satisfaciendo algunas restricciones de suavizado (no puede cambiar más de 1º el ángulo de slant entre cada columna). Finalmente, la mejor secuencia en esa matriz nos da el ángulo que hay que aplicar en cada columna de la imagen. La figura 8.3 ilustra un ejemplo de aplicación de esta idea, y podemos ver el resultado en la figura 8.2(f).

8.2.4. Normalización del tamaño

Una vez se ha eliminado el slope y el slant, el tamaño de la línea de texto se normaliza para minimizar la variaciones en tamaño y posición de las tres áreas (cuerpo central, ascendentes, descendentes). Es más, se reduce el tamaño de los ascendentes y descendentes respecto al cuerpo central, dado que aportan una información menos importante por encontrarse en blanco en la gran mayoría de los casos. [1]

La forma más natural de normalizar el tamaño consiste en detectar las líneas de referencia, y normalizar el texto respecto a ellas [Bengio *et al.*, 1995; Simard *et al.*, 2005; Gorbe *et al.*, 2008; Schenk *et al.*, 2008]. Siguiendo las ideas previas, se detectan los extremos locales extraídos con el método de la sección 8.2.1 y se clasifican mediante un MLP. En esta ocasión tenemos 5 clases diferentes: las cuatro líneas de referencia, y una clase más que agrupa todos los

[1] Preservamos la presencia o ausencia de ascendentes/descendentes, así como su ancho, pero no su altura.



Original image.

Preprocessing result.

Feature extraction result.

Figura 8.4: *Ejemplo del preproceso y la extracción de características sobre una línea de la IAM-DB [España-Boquera* et al.*, 2011].* // Preprocess and feature extraction example.

puntos que no pertenecen a ninguna de las líneas. A este MLP le llamaremos Normalize-MLP. Los puntos pertenecientes a la misma clase se usan para ajustar por interpolación lineal las líneas de referencia (véase la figura 8.2(g)). Estas líneas definen las tres áreas a normalizar. La normalización se aplica columna a columna, haciendo un escalado lineal de las tres zonas a una altura fija. Los ascendentes se reducen para ocupar un 20 % del alto de la imagen final, y los descendentes al 10 %. La figura 8.2(h) ilustra un ejemplo de imagen normalizada.

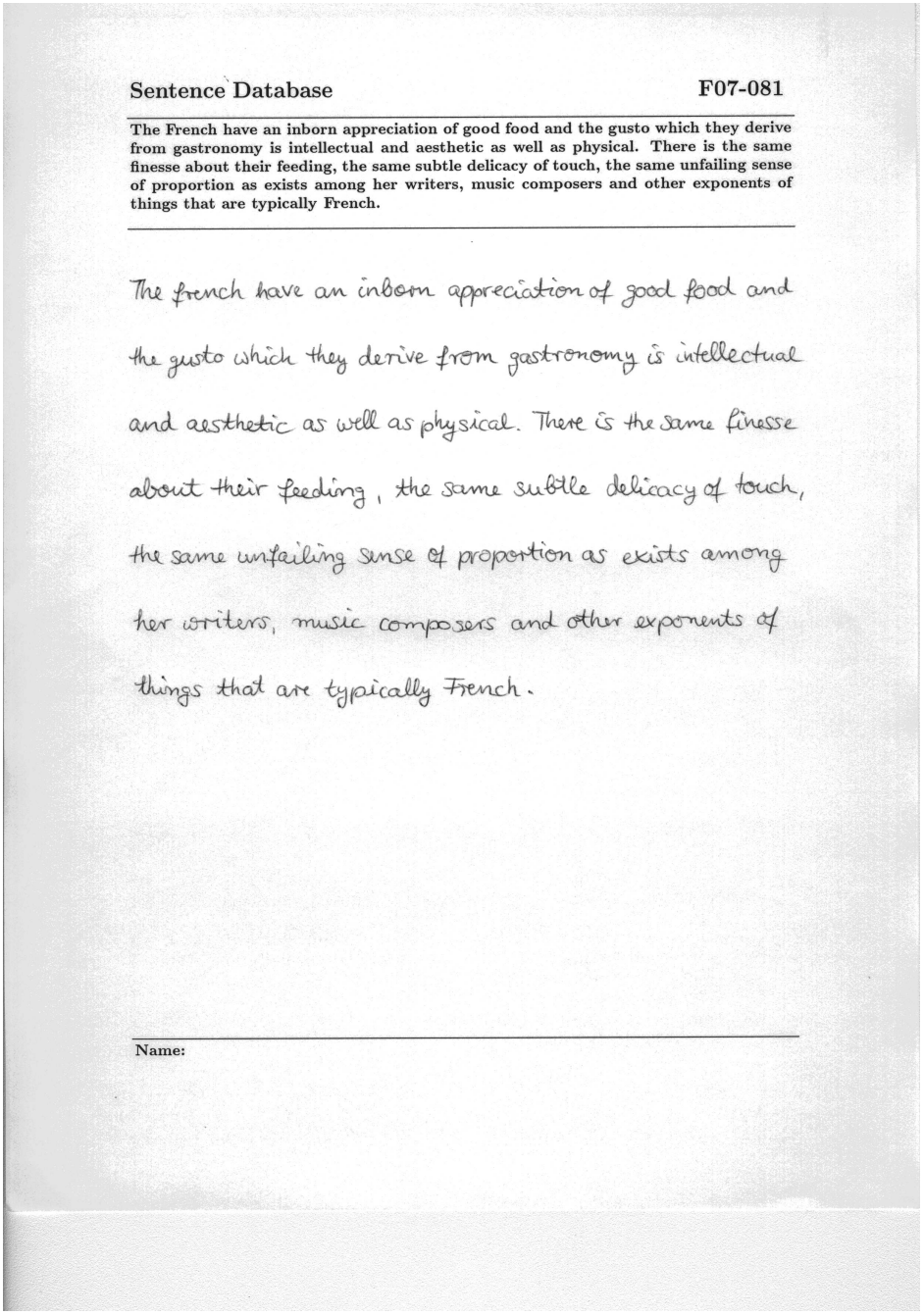
8.2.5. Extracción de características

Tras todo el preproceso, es necesario aplicar un método de extracción de características, para capturar la información más relevante de cada carácter a reconocer. En el sistema utilizado, cada imagen se convierte en una secuencia de vectores de características, siguiendo a [Toselli *et al.*, 2004]. Las derivadas horizontal y vertical de la imagen se añaden para formar parte del vector de características, así como una versión suavizada y más compacta de la imagen original. En suma, de cada columna de píxeles se extraen 60 características finales. La figura 8.4 muestra un ejemplo de línea extraída de la IAM-DB [Marti & Bunke, 2002], y el resultado de aplicar el preproceso y la extracción de características final.

8.3 Corpus IAM-DB

Toda la experimentación se ha realizado sobre el corpus IAM-DB [Marti & Bunke, 2002]. El corpus de escritura manuscrita IAM-DB [Marti & Bunke, 2002] (versión 3.0) consiste en 1 500 formularios de texto manuscrito escaneados, procedentes de 650 escritores diferentes (las figuras 8.5 y 8.6 ilustran dos ejemplos de formularios escaneados). En total se tienen 13 000 imágenes de líneas de texto manuscrito, sin restricción de estilo de escritura ni de instrumento estilográfico. Los textos escritos están extraídos del corpus Lancaster-Oslo/Bergen (LOB) [Johansson *et al.*, 1986].

Existe una tarea estándar [Graves *et al.*, 2009; Bertolami & Bunke, 2008a,b], independiente del escritor, consistente en 6 161 líneas (283 escritores) para entrenar, 920 líneas (56 escritores) para validación y 2 781 líneas (161 escritores) para test. Todos estos conjuntos son disjuntos y los escritores no se repiten entre los conjuntos. La figura 8.7 muestra algunas



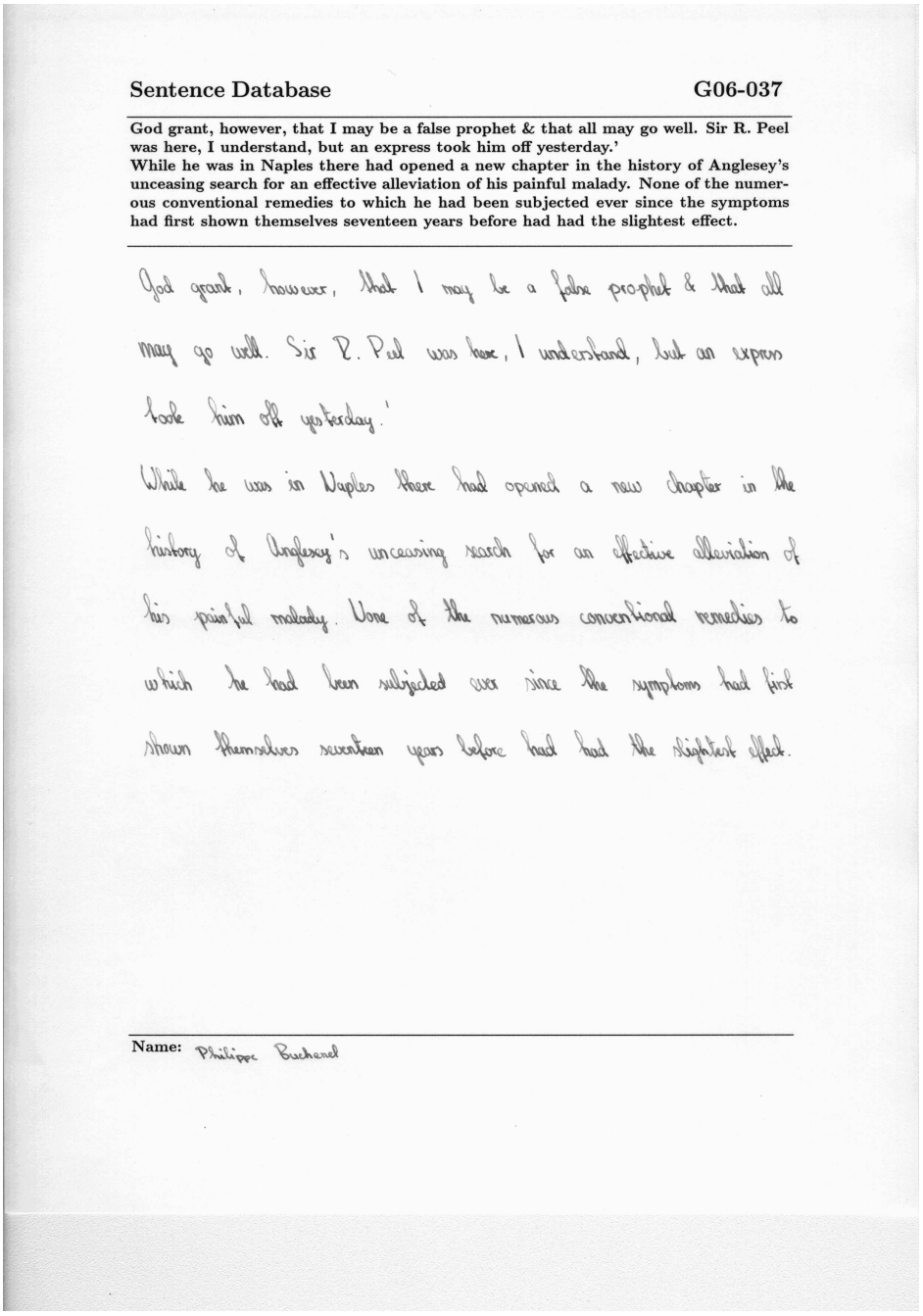
Sentence Database F07-081

The French have an inborn appreciation of good food and the gusto which they derive from gastronomy is intellectual and aesthetic as well as physical. There is the same finesse about their feeding, the same subtle delicacy of touch, the same unfailing sense of proportion as exists among her writers, music composers and other exponents of things that are typically French.

The french have an inborn appreciation of good food and the gusto which they derive from gastronomy is intellectual and aesthetic as well as physical. There is the same finesse about their feeding, the same subtle delicacy of touch, the same unfailing sense of proportion as exists among her writers, music composers and other exponents of things that are typically French.

Name:

Figura 8.5: *Imagen de un formulario escaneado para la tarea de escritura manuscrita IAM-DB. //* IAM-DB document image.



Sentence Database G06-037

God grant, however, that I may be a false prophet & that all may go well. Sir R. Peel was here, I understand, but an express took him off yesterday.'
While he was in Naples there had opened a new chapter in the history of Anglesey's unceasing search for an effective alleviation of his painful malady. None of the numerous conventional remedies to which he had been subjected ever since the symptoms had first shown themselves seventeen years before had had the slightest effect.

God grant, however, that I may be a false prophet & that all may go well. Sir R. Peel was here, I understand, but an express took him off yesterday.'

While he was in Naples there had opened a new chapter in the history of Anglesey's unceasing search for an effective alleviation of his painful malady. None of the numerous conventional remedies to which he had been subjected ever since the symptoms had first shown themselves seventeen years before had had the slightest effect.

Name: Philippe Buchenel

Figura 8.6: *Otra imagen de un formulario escaneado para la tarea de escritura manuscrita IAM-DB.* // Another IAM-DB document image.

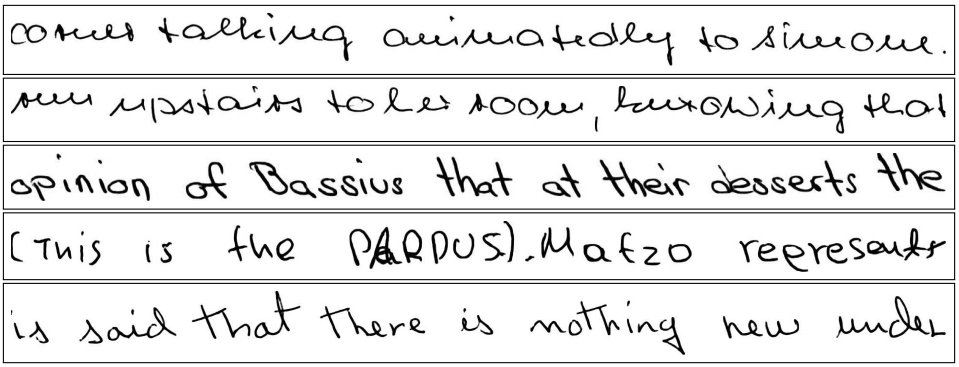


Figura 8.7: *Ejemplos de líneas extraídas del corpus IAM-DB. //* Some line images from the IAM-DB corpus.

Tabla 8.1: *Grafemas de la IAM-DB. //* IAM-DB character set.

Lowercase	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
Uppercase	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	<space>- , ; : ! ? / . ' "( ) * & # +

líneas del corpus. Hay un total de 87 967 instancias de 11 320 palabras distintas en la unión de los tres conjuntos.

El léxico es modelado con 80 caracteres: 26 letras minúsculas, 26 mayúsculas, 10 dígitos, 16 signos de puntuación, el espacio blanco y un carácter que representa los tachones. Una lista completa de estos grafemas se detalla en la tabla 8.1.

8.4 Entrenamiento de los modelos ópticos

Se han entrenado modelos ópticos HMM/ANN mediante un algoritmo de Expectación-Maximización (EM) basado en Viterbi. La red neuronal se ha entrenado normalizando a media cero y desviación típica de cada una de las características de los datos de entrada, añadiéndoles además un pequeño ruido Gaussiano. La topología de los modelos de Markov es de 7 estados, con bucle y transición al siguiente estado. La tabla 8.2 muestra los parámetros del algoritmo de entrenamiento y de la red utilizada.

8.5 Material para el modelo de lenguaje: corpus LIBW

Para el modelo de lenguaje se han utilizado los corpus textuales LOB [Johansson *et al.*, 1986] (excluyendo del mismo las frases que forman parte de la validación y del test de la IAM-DB), Brown [Francis & Kucera, 1979] y Wellington [Bauer, 1993]. Llamaremos a la unión de todo este material textual corpus “LIBW”.

Tabla 8.2: *Parámetros de los modelos ópticos HMM/ANN. //* HMM/ANN models parameters.

Replacement:	$300K$ training.
	$200K$ validation.
# of graphemes:	8
# of states per grapheme:	7
Frame size:	60
NN context:	4 left
	current frame
	4 right
NN topology:	$(4 + 1 + 4) \cdot 60 - 192 - 128 - 7 \cdot 80$
NN initial weights range:	$[-0.7, 0.7]$
Learning Rate:	0.001
Momentum:	0.005
Weight Decay:	1×10^{-10}
Gaussian noise:	$(\mu = 0, \sigma = 0.015)$

Tabla 8.3: *Estadísticas del corpus LIBW para entrenar el modelo de lenguaje y de las particiones de validación y test de la IAM-DB. //* Textual corpora statistics for the IAM-DB task. Referenced at page 308.

Corpora	# lines	# words	# characters	% out-of-vocabulary words (OOV)
LOB + IAM Training	174K	2.3M	11M	–
Brown	114K	1.1M	12M	–
Wellington	114K	1.1M	11M	–
Total	402K	4.5M	34M	–
IAM Validation	920	8 762	41 104	3.18 %
IAM Test	2 781	25 424	123 405	2.86 %

El diccionario utilizado consta de las 103K palabras que aparecen en la concatenación de los 3 corpus. Se diferencian mayúsculas de minúsculas, y se mantienen los signos de puntuación. La tabla 8.3 resume las estadísticas del material utilizado.

8.6 Reconociendo líneas de texto

Cuando estamos trabajando en HTR, el reconocimiento se realiza a nivel de línea de texto, tal y como se ilustra en los ejemplos de la figura 8.7. A diferencia de la aproximación al ASR, donde se reconocen frases, las líneas no tienen un inicio y un fin de frase. Una modificación sencilla de los algoritmos permite dar cuenta de este detalle, permitiendo lanzar el reconocimiento sobre líneas donde el comienzo puede ser cualquier punto en una frase, y su final también puede recaer sobre cualquier palabra. La aproximación consiste en modificar la ecuación (2.2) de los N-gramas donde habíamos dado cuenta del comienzo y fin de frase mediante el bcc y el ecc. Lo que haremos será, en lugar de forzar a que la historia inicial del modelo de lenguaje sea el bcc, forzaremos a que la historia inicial esté vacía, con lo que la primera transición que haremos en el modelo de lenguaje será la correspondiente al

unigrama, la siguiente transición será la del bigrama, y así sucesivamente transitando a través del modelo de estados finitos. La última de las transiciones se forzaba a que fuera con la etiqueta ecc, ahora eliminaremos dicha transición del cómputo de la probabilidad.

Veamos un ejemplo: dada la secuencia $\omega_1\omega_2\omega_3\omega_4$, y un modelo de lenguaje de 3-gramas, la probabilidad se calculará como este productorio:

$$p(\omega_1\omega_2\omega_3\omega_4) = p(\omega_1)p(\omega_2|\omega_1)p(\omega_3|\omega_1\omega_2)p(\omega_4|\omega_2\omega_3)\,, \tag{8.1}$$

eliminando del cómputo a $p(\omega_1|bcc)$ y a $p(ecc|\omega_3\omega_4)$. Esta simple modificación permite que el algoritmo de búsqueda sea más flexible y cometa menos errores en el comienzo y final de las líneas de texto cuando estas no son frases completas.

Otra forma de aliviar este problema sería generalizar el proceso de reconocimiento, pasando de reconocer líneas a reconocer párrafos. No obstante hay que ser cuidadoso para llevar a cabo este salto, pues conlleva un incremento en el tamaño de la entrada $\bar{x}$ e implica un incremento del coste computacional. Quizá podrían utilizarse técnicas como el "Partial Backtracing" que permiten generar la salida del sistema a trozos, evitando tener que esperar hasta que termine la decodificación completa.

8.6.1. Extensión a NN LMs

El problema que tiene la ecuación (8.1) cuando se utilizan NN LMs es consecuencia precisamente de las ventajas de estos modelos. Los NN LMs computan la probabilidad condicional de una palabra dada la historia completa de las $N-1$ palabras anteriores, ya que interpolan las entradas no vistas durante el entrenamiento. Esto acarrea el problema de que si se elimina el bcc, ¿cuál debería ser la entrada de la red neuronal para que calcule la probabilidad condicional?

Podemos resolverlo de diversas formas, algunas de ellas serían:

- Utilizar un modelo estándar de N-gramas sólo para las $N-1$ primeras palabras, ya que no disponemos de contexto para la red neuronal. Esta idea se puede combinar con smoothed Fast NN LMs, utilizando un unigrama estándar para la primera palabra, y a partir de esta utilizar el bigrama conexionista, después el trigrama, ...

- Estimar de alguna manera una codificación para una palabra *ficticia* que represente el evento de *no hay contexto*. Esta palabra se podría codificar de muchas formas. Quizá la más simple sería hacer una media ponderada de la codificación distribuida de las palabras del vocabulario, algo así como calcular el centro de masas de la codificación en el espacio continuo que utilizan los NN LMs utilizando las cuentas de unigramas del conjunto de entrenamiento o algún otro heurístico como ponderación.

- Entrenar los modelos con frases segmentadas en líneas mediante algún tipo de heurístico. Esta opción permite dotar de flexibilidad al modelo de lenguaje a través de la creación de puntos de comienzo y fin de frase aleatorios.

De estas ideas en este trabajo hemos probado la primera y la tercera, combinadas con la idea de utilizar smoothed Fast NN LMs.

Por otra parte, eliminar el ecc en la ecuación no plantea ningún problema, ya que es una transición extra que se añade al final del algoritmo de Viterbi para poder estimar la probabilidad de que una palabra sea final de frase. Basta con eliminar dicha transición del cómputo.

8.7 Experimentos con modelos de palabras

Esta sección describe la experimentación realizada para comparar los NN LMs con los modelos de lenguaje estándar. Se presentan diferentes configuraciones de parámetros de NN LMs, pudiéndose observar una mejora robusta a nivel de WER.

8.7.1. Configuración de los modelos de lenguaje utilizados

Para comparar resultados se han entrenado bigramas, 3-gramas y 4-gramas, mediante el SRI tookit [Stolcke, 2002], configurado para generar un modelo interpolado con el descuento de Kneser-Ney modificado (mKN), que es el que mejores resultados obtiene.

Cada NN LM es la combinación del modelo de bigramas mKN y cuatro redes neuronales que cambian en la semilla inicial del generador de números pseudo-aleatorios, y en el tamaño de la codificación distribuida de cada palabra (160, 192, 224, 256). Todas las redes neuronales tienen 200 unidades en la capa oculta, y una Short-List de las 10K palabras más frecuentes en la salida de la red neuronal, si bien se ha utilizado la ecuación (3.5) para extender la Short-List al vocabulario completo de la tarea, formado por 103K palabras. Se han probado diferentes configuraciones para la entrada de la red neuronal, dependiendo de un umbral Θ, de manera que para el vocabulario de entrada Ω^I de la red neuronal se usan todas las palabras cuya frecuencia absoluta de aparición sea estrictamente mayor que Θ. Se han probado valores de $\Theta = 21, 10, 8, 1$, que se corresponden con diccionarios de tamaño 10K, 17K, 19K, 56K, respectivamente.

Con todos estos parámetros se han entrenado bigramas, 3-gramas y 4-gramas de los siguientes modelos de lenguaje:

- **20K mKN**: a efectos de comparar nuestro sistema de reconocimiento con el descrito en [Graves *et al.*, 2009] hemos utilizado exactamente el mismo modelo de lenguaje de bigramas que en aquel trabajo, que utiliza únicamente las 20K palabras más frecuentes.

- **mKN**: modelo estándar de N-gramas entrenado con todo el vocabulario usando la herramienta SRI [Stolcke, 2002], configurado para usar la técnica de suavizado Kneser-Ney modificado, y configurado para crear un modelo interpolado. Será nuestro baseline, y ha sido creado entrenando un modelo de lenguaje para cada conjunto de entrenamiento posible (LOB, Wellington, Brown), y luego interpolando los 3 modelos para minimizar la PPL en el conjunto de validación.

- $\Theta = v$ **NN**: NN LM estándar, entrenado mediante la técnica con reemplazamiento sobre todo el conjunto de entrenamiento. Se usa la técnica de precalcular constantes de normalización softmax para acelerar los cálculos, pero cuando una constante no se encuentra, esta se *calcula al vuelo*. El valor v indica la frecuencia absoluta de corte, es decir, el valor de Θ, tal y como se ha explicado anteriormente. El resto de parámetros es el mismo que se ha utilizado antes.

- $\Theta = v$ **FNN**: smoothed Fast NN LM, entrenado siguiendo la misma técnica con reemplazamiento, y de nuevo sobre todo el conjunto de entrenamiento. En este caso se usa la técnica de aceleración, pero cuando una constante de normalización softmax no se encuentra precalculada, se utiliza el siguiente modelo más simple. Por lo tanto, tenemos tres posibles configuraciones:

Tabla 8.4: *Perplejidad con el conjunto de validación del corpus LIBW. //* Validation set perplexity. Referenced at pages 308 and 309 .

System	$\|\Omega^I\|$	bigram	3-gram	4-gram
mKN	103K	**659**	664	671
$\Theta = 21$ NN	10K	527	525	**515**
$\Theta = 10$ NN	17K	535	528	534
$\Theta = 8$ NN	19K	543	563	611
$\Theta = 1$ NN	56K	538	566	617
$\Theta = 21$ FNN	10K	527	496	**479**
$\Theta = 10$ FNN	17K	535	497	491
$\Theta = 8$ FNN	19K	543	503	493
$\Theta = 1$ FNN	56K	538	505	497

1. sólo bigramas;
2. 3-gramas y si no, bigramas;
3. 4-gramas; si no, 3-gramas y, en último lugar, bigramas.

Análisis de la PPL

La tabla 8.4 muestra los resultados de PPL con el conjunto de validación del corpus LIBW para cada uno de los modelos propuestos. El mejor resultado de PPL para **mKN** corresponde al bigrama, mientras que para **NN** y **FNN** es el 4-grama con $\Theta = 21$, es decir, con el vocabulario de menor tamaño para la entrada de la red neuronal. También ilustra el efecto de usar la técnica de aceleración para smoothed Fast NN LMs, que mejora la perplejidad en $\approx 7\,\%$ (36 puntos absolutos) para el 4-grama, siendo este un resultado únicamente extrapolable a tareas de reconocimiento de líneas de texto. Esto sucede debido a que no es conocido el contexto inicial de las palabras de cada línea. Dada esta situación, el modelo **NN** sólo utiliza el NN LM a partir de la cuarta palabra de la línea, utilizando para todas las anteriores el bigrama **mKN**. Esta es la razón también por la que para la mayoría de los casos de **NN** el 4-grama obtiene peor PPL que el 3-grama.

La mejora respecto al sistema **mKN** oscila entre $\approx\ 6\,\%$ (42 puntos absolutos, si comparamos 659 con 617) y $\approx\ 27\,\%$ (180 puntos absolutos, si comparamos 659 con 479). Las diferencia en PPL entre los distintos NN LMs probados (estándar NN LMs y smoothed Fast NN LMs, con distintos tamaños de vocabulario de entrada) muestran una fuerte correlación entre el tamaño del vocabulario de entrada y el valor N del N-grama para los sistemas **NN**. Al aumentar el vocabulario y aumentar el valor de N aumenta la PPL de forma considerable. Si nos fijamos en los sistemas **FNN** las diferencias entre 3-gramas y 4-gramas son poco significativas. Si comparamos bigramas con 3-gramas, o 4-gramas, sí que encontramos diferencias significativas.

8.7.2. Análisis de los resultados

En la tabla 8.5 se encuentran los resultados de la experimentación de reconocimiento de escritura para la tarea IAM-DB sobre el conjunto de validación, y en la tabla 8.6 los resultados con el conjunto de test. Vamos a analizar los resultados por partes, comparando

Tabla 8.5: *Resultados de WER /CER /SER con el conjunto de validación de la IAM-DB utilizando distintos modelos de lenguaje. //* WER /CER /SER for the validation set. Referenced at page 308.

System	% WER / % CER / % SER in validation bigram	3-gram	4-gram
20K mKN	20.1 / 7.4 / 73.2	–	–
mKN	**17.3 / 6.2** / 69.8	17.8 / 6.3 / 70.3	17.9 / 6.3 / **69.3**
Rescoring with NN LMs			
$\Theta = 21$ NN	16.0 / 5.8 / 67.4	16.0 / 5.9 / 67.2	16.2 / 5.8 / 67.2
$\Theta = 10$ NN	15.9 / 5.9 / 67.5	16.0 / 5.9 / 67.2	16.4 / 5.9 / 66.5
$\Theta = 8$ NN	16.0 / 5.8 / 66.9	16.3 / 5.9 / 67.8	16.9 / 6.1 / 68.8
$\Theta = 1$ NN	16.0 / 5.8 / 66.6	16.3 / 6.0 / 67.9	16.9 / 6.2 / 69.2
$\Theta = 21$ FNN	16.0 / 5.8 / 67.4	15.8 / **5.7** / 66.0	15.8 / **5.7** / 65.1
$\Theta = 10$ FNN	15.9 / 5.9 / 67.5	15.9 / **5.7 / 65.0**	15.9 / 5.8 / 66.0
$\Theta = 8$ FNN	16.0 / 5.8 / 66.9	15.8 / 5.8 / 65.8	15.8 / **5.7** / 65.8
$\Theta = 1$ FNN	16.0 / 5.8 / 66.6	15.9 / **5.7** / 66.1	**15.7** / 5.8 / 66.3
NN LMs integrated during decoding			
$\Theta = 21$ NN	16.0 / 5.8 / 67.0	16.0 / 5.8 / 67.2	16.1 / 5.8 / 67.2
$\Theta = 10$ NN	16.1 / 5.8 / 66.9	16.1 / 5.8 / 67.3	16.4 / 5.8 / 66.7
$\Theta = 8$ NN	16.1 / 5.9 / 67.0	16.3 / 5.9 / 67.9	16.9 / 6.0 / 69.2
$\Theta = 1$ NN	16.1 / 5.8 / 66.7	16.4 / 6.0 / 67.5	16.8 / 6.2 / 69.3
$\Theta = 21$ FNN	16.0 / 5.8 / 67.0	**15.8** / 5.8 / 66.5	**15.8 / 5.6** / 65.2
$\Theta = 10$ FNN	16.1 / 5.8 / 66.9	15.9 / 5.7 / **65.0**	16.0 / 5.8 / 65.6
$\Theta = 8$ FNN	16.1 / 5.9 / 67.0	**15.8** / 5.8 / 65.7	**15.8** / 5.7 / 65.6
$\Theta = 1$ FNN	16.1 / 5.8 / 66.7	15.9 / 5.7 / 66.1	**15.8** / 5.7 / 65.6

Tabla 8.6: *Resultados de WER /CER /SER con el conjunto de test de la IAM-DB para los mejores modelos de lenguaje. El intervalo de confianza es del* 95 %. *//* WER /CER /SER for the test set. Referenced at pages 308 and 309 .

System	Test % WER	% CER	% SER
[Graves *et al.*, 2009]	25.9 ± 0.8	–	–
bigram 20K mKN	23.4 ± 0.8	9.6 ± 0.4	78.1 ± 1.6
bigram mKN	21.9 ± 0.7	8.8 ± 0.4	76.0 ± 1.6
Rescoring with NN LMs			
4-gram $\Theta = 21$ FNN	$\mathbf{20.2 \pm 0.7}$	$\mathbf{8.3 \pm 0.4}$	$\mathbf{72.9 \pm 1.6}$
NN LMs integrated during decoding			
4-gram $\Theta = 21$ FNN	$\mathbf{20.2 \pm 0.7}$	$\mathbf{8.3 \pm 0.4}$	73.0 ± 1.6

la calidad de nuestro baseline (**mKN**) con el estado del arte, y más tarde comparando los resultados obtenidos con NN LMs haciendo rescoring de listas de N-best e integrando de forma totalmente acoplada los modelos en la búsqueda.

Análisis del baseline frente al estado del arte

Observando los resultados de validación, los modelos **mKN** tienen un comportamiento claro, no es posible mejorar usando valores de N más allá de 2 (bigramas). A efectos comparativos, se ha calculado el error en reconocimiento usando exactamente el mismo modelo de lenguaje que [Graves *et al.*, 2009] (**20K mKN**). Observamos que con el conjunto de validación el modelo de bigramas **mKN** con todo el vocabulario obtiene una mejora de $\approx 14\,\%$ de WER (2.8 puntos absolutos), $\approx 16\,\%$ de CER (1.2 puntos absolutos) y $\approx 4.6\,\%$ de SER (3.4 puntos absolutos). Estas mejoras son debidas a un reentrenamiento del modelo utilizando el vocabulario completo (103K), en lugar de las 20K palabras más frecuentes.

En test siguen observándose mejoras del bigrama **mKN** respecto al **20K mKN**. Se observa una mejora en WER de $\approx 6.4\,\%$ (1.5 puntos absolutos), $\approx 8.3\,\%$ de CER (0.8 puntos absolutos) y $\approx 2.6\,\%$ de SER (2.1 puntos absolutos). Si observamos los intervalos de confianza, la diferencia en WER y CER es estadísticamente significativa, mientras que en SER no lo es.

Es posible comparar el resultado en test obtenido en [Graves *et al.*, 2009] y con nuestro sistema usando exactamente el mismo modelo de lenguaje (**20K mKN**) y usando el modelo **mKN**. La mejora en WER es de un $\approx 9.7\,\%$ (2.5 puntos absolutos) y $\approx 15.4\,\%$ (4 puntos absolutos) respectivamente. La mejora es claramente significativa en ambos casos.

Tras este análisis podemos concluir que el sistema *mejora de forma significativa* los mejores resultados publicados hasta el momento para esta tarea. Esto quiere decir que el reto de los NN LMs en esta tarea es aún mayor, ya que cuanto mejor es un sistema, más difícil es reducir su error.

Análisis del tamaño del vocabulario de entrada de los NN LMs

Se ha experimentado con 4 tamaños del vocabulario Ω^I de la entrada de los NN LMs: $\Theta = 21, 10, 8, 1$, donde Θ es el umbral de frecuencia absoluta de corte, es decir, Ω^I comprende todas las palabras que aparecen más de Θ veces en el conjunto de entrenamiento. Cada uno de estos umbrales se corresponde con la fracción de vocabulario de las 10K, 17K, 19K y 59K palabras más frecuentes.

Para este análisis vamos a centrarnos en los modelos **FNN**. En la tabla 8.4 podemos observar las diferencias en PPL entre los NN LMs entrenados para cada valor de Θ y para cada valor de $N = 2, 3, 4$. Se observa que hay una clara tendencia a aumentar la PPL conforme aumenta la talla de Ω^I.

Esta tendencia no se observa en la tabla 8.5, a nivel de resultados de reconocimiento, si bien se puede observar que las diferencias en WER, CER y SER no son significativas en ningún caso, aunque en SER sí que parece, en general, observarse un pequeño empeoramiento sistemático de los resultados al aumentar el vocabulario.

Encontramos una clara ventaja de los NN LMs frente al sistema **mKN**, una mejora entre $\approx 7.5\,\%$ y $\approx 9.2\,\%$ de WER (1.3 y 1.6 puntos absolutos) si comparamos el peor **FNN** (16.0 de WER) y el mejor **FNN** (15.7 de WER) con el bigrama **mKN** (17.3 de WER).

Tabla 8.7: *Porcentaje de palabras acertadas con el conjunto de test de la tarea IAM-DB, desglosado en la primera, la última, y el resto de palabras de cada línea, para 4-gramas de las configuraciones **NN** y **FNN** integrados en la decodificación. //* Word accuracy for the test set focusing only in the first word, last word and the rest. Referenced at page 309.

System	% accuracy First word	Last word	Rest
4-gram $\Theta = 21$ NN	75.9	78.5	83.2
4-gram $\Theta = 10$ NN	76.0	78.5	83.5
4-gram $\Theta = 8$ NN	74.6	78.2	82.5
4-gram $\Theta = 1$ NN	74.8	78.3	82.1
4-gram $\Theta = 21$ FNN	78.0	78.4	83.6
4-gram $\Theta = 10$ FNN	78.1	78.6	83.3
4-gram $\Theta = 8$ FNN	78.0	78.3	83.1
4-gram $\Theta = 1$ FNN	78.2	78.4	83.2

Comparando la estrategia de NN LMs estándar (NN) con NN LMs acelerados (FNN)

Si seguimos analizando los resultados de validación para **NN** y **FNN**, se observa un resultado a priori sorprendente. El sistema **FNN**, que utiliza las técnicas de evaluación rápida del capítulo 4 y por tanto utiliza un sistema *degradado* en calidad, obtiene mejores resultados que los NN LMs estándar. Sin embargo, esta mejoría es fácilmente explicable si nos situamos en el marco del reconocimiento de líneas en los sistemas de escritura manuscrita (véase la sección 8.6). Dentro de dicho marco la estrategia estándar de NN LMs tiene problemas con el comienzo de las líneas. Esto es así debido a que un NN LM para 4-gramas precisa de tres palabras en su contexto, lo cual sucede únicamente a partir de la cuarta palabra reconocida.

Esta característica está presente únicamente en los sistemas de reconocimiento de líneas, y no es extrapolable a otras tareas.[2]. Para demostrar la hipótesis de que la diferencia entre los sistemas **NN** y **FNN** se debe a la idiosincrasia de la tarea de reconocimiento de líneas, se ha calculado el porcentaje de palabras acertadas diferenciando las frases en tres partes: la primera palabra, la última palabra y el resto de palabras. La tabla 8.7 muestra estos resultados. El porcentaje de aciertos en la primera palabra es entre dos y tres puntos *más alto* para el sistema **FNN**, mientras que para la última palabra y el resto las diferencias no son significativas. Esta diferencia se hace más acusada conforme aumenta el tamaño de Ω^I, es decir, conforme se reduce el umbral Θ. Hay que tener en cuenta que el número medio de palabras por línea en la tarea IAM-DB es de 9 palabras, por lo que, cuando se utilizan 4-gramas, casi la mitad de las palabras de la frase se reconocen utilizando un modelo de lenguaje de peor calidad.

Comparando el rescoring de listas de N-best frente al sistema integrado

No hay diferencias significativas entre ambas estrategias, ni en validación ni en test. En ambos casos es ligeramente mejor la estrategia **FNN**, y no existen diferencias significativas entre los distintos valores de Θ para el vocabulario Ω^I. Para profundizar en el análisis, se ha estimado el WER entre la salida del sistema **FNN** integrado y haciendo rescoring, y el WER entre cada uno de ellos y el baseline **mKN** (véase la tabla 8.8) con el propósito de

[2]Una solución a este problema sería diseñar el sistema de reconocimiento para que funcione a nivel de documento o de párrafo, de manera que el modelo de lenguaje tenga siempre disponible el contexto inicial de las palabras.

Tabla 8.8: *WER y precisión (%) comparando el sistema que integra NN LMs con el sistema que hace rescoring de listas de N-best, y también con el baseline estándar. Se ha escogido el parámetro $\Theta = 21$.* // WER and accuracy for integrated and rescoring systems using input vocabulary $\Theta = 21$.

Compared systems			% WER	% Accuracy
4-gram FNN integrated	vs	4-gram FNN rescoring	0.2	99.9
4-gram FNN rescoring	vs	bigram mKN	6.9	93.7
4-gram FNN integrated	vs	bigram mKN	7.0	93.6

comparar las palabras que han generado error entre los distintos sistemas. Se ha representado tanto el WER como el porcentaje de precisión (accuracy)[3] entre ambos sistemas. Se observa que, entre el sistema integrado y el que funciona mediante rescoring de listas de N-best, las diferencias son despreciables ($99.9\,\%$ de precisión y $0.2\,\%$ de WER). Si comparamos ambos sistemas con el baseline, se observa que el sistema integrado presenta una precisión del $93.6\,\%$ y un WER del $7.0\,\%$, mientras que el sistema que hace rescoring de listas de N-best tiene una precisión del $93.7\,\%$ y un WER del $6.9\,\%$. A partir de estos resultados, se puede afirmar que las diferencias entre ambos sistemas son muy poco significativas. Por otro lado, el sistema que hace rescoring de listas de N-best presenta mayor parecido con el baseline **mKN**, siendo este resultado no significativo.

Análisis de los resultados de test con NN LMs

El reconocimiento del conjunto de test se ha realizado con el modelo 4-grama $\Theta = 21$ **FNN**, y se ha comparado la estrategia integrada con la de rescoring de listas de N-best. Este es el modelo 4-grama cuyo tamaño de Ω^I es el más reducido, y utiliza las técnicas de evaluación rápida de los NN LMs. Ha sido escogido porque las diferencias entre los distintos valores de Θ no eran significativas y la estrategia **NN** daba peores resultados en validación.

Comparándolo todo, la tabla 8.6 muestra una clara *ventaja* al usar NN LMs frente a no usarlos, obteniendo una mejora de $\approx 7.7\,\%$ de WER (1.7 puntos absolutos), $\approx 5.7\,\%$ de CER (0.5 puntos absolutos) y $\approx 4.1\,\%$ de SER (3.1 puntos absolutos). Las diferencias en WER son significativas, para los intervalos de confianza del $95\,\%$, ya que el intervalo deja de solaparse si la diferencia es mayor a 1.4 puntos absolutos, y en este caso es de 1.7 puntos. En SER los intervalos están poco solapados, ya que se establece que es necesario una diferencia de 3.2 puntos absolutos para que dejen de solaparse, y se ha obtenido una diferencia de 3.1 puntos.

8.7.3. Discusión

Esta sección ha presentado los resultados de aplicar las técnicas de modelado conexionista de lenguaje desarrolladas en la primera parte de esta tesis a una tarea de escritura manuscrita. La comparación con otros sistemas siempre es muy difícil, por ello se ha optado por utilizar la base de datos IAM-DB, de uso estándar en escritura manuscrita. También hemos comparado los resultados presentados con el sistema HMM/ANN y el modelo de lenguaje de [Graves *et al.*, 2009], pudiendo así comparar las mejoras obtenidas de forma justa.

Hemos observado que para tareas de reconocimiento de líneas, los NN LMs presentados en esta tesis, utilizando la técnica de evaluación rápida, tienen ventajas frente a los NN LMs

[3]Medimos esta precisión como el porcentaje de palabras alineando la salida de ambos sistemas mediante programación dinámica y contando el número de palabras comunes.

estándar. Comparando diferentes tallas de vocabulario de entrada se ha encontrado que este parámetro no afecta en esta tarea. No obstante, esta conclusión no es extrapolable a otras tareas, ya que nuestra hipótesis es que las características concretas de la tarea IAM-DB causan este comportamiento (líneas con pocas palabras). En otros corpus sí que hemos encontrado algunas mejoras al aumentar la talla del vocabulario de entrada de la red neuronal (véase sección 3.1.3), si bien las diferencias no son significativas. Es un tema de investigación abierto encontrar formas de mejorar la codificación en el espacio continuo de las palabras, ya que una mejor comprensión de esta codificación puede lograr explicar este comportamiento.

Finalmente el resultado obtenido en WER ($20.2\,\% \pm 0.7$) y CER ($8.3\,\% \pm 0.4$) es, hasta donde sabemos, el *mejor resultado* existente para la tarea presentada con esta base de datos. Los resultados son estadísticamente significativos para una confianza del $95\,\%$.

8.8 Experimentos con modelos de grafemas

Esta sección presenta los experimentos realizados usando modelos de lenguaje donde la unidad lingüística son grafemas, en lugar de palabras. La ventaja más importante de este cambio de concepto es doble: el número de unidades (grafemas) es órdenes de magnitud menor, y aunque no hay un vocabulario explícito, el modelo puede aprender la forma en que son construidas las palabras a partir de la concatenación de grafemas. Esto permite que el modelo sea capaz de reconocer palabras que en el conjunto de entrenamiento no existían. Por tanto, uno de los resultados más interesantes de esta sección es la medida del error del modelo de grafemas para reconocer dichas palabras desconocidas. Obviamente utilizaremos NN LMs y contrastaremos sus resultados con modelos de lenguaje estándar.

8.8.1. Modelos de grafemas

Se han realizado experimentos sobre el corpus IAM-DB, entrenando NN LMs a nivel de grafema , es decir, en lugar de usarlos para estimar la probabilidad de una hipótesis descomponiéndola en palabras, se han usado para estimar dicha probabilidad al nivel de cada uno de los grafemas que aparecen en la hipótesis, incluyendo los espacios en blanco. El material de entrenamiento de estos modelos de lenguaje es ligeramente diferente al utilizado en la sección anterior. El origen de los datos es el mismo (corpus LIBW), pero en este caso se ha segmentado el texto en líneas siguiendo un heurístico aleatorio con el fin de mitigar el problema de la mala calidad de reconocimiento en el comienzo y el final de las líneas de texto. Además, se ha utilizado la aproximación smoothed Fast NN LM descrita en el capítulo 4, permitiendo a los modelos conexionistas utilizar un modelo de menor orden en caso de no encontrar la constante de normalización correspondiente. Esto también permite hacer frente a la idiosincrasia del reconocimiento de líneas.

El modelo de lenguaje se basa en el mismo principio que los modelos de lenguaje basados en palabras. Para estos experimentos, hemos combinado el modelo conexionista con un modelo estadístico estándar de N-gramas siguiendo esta fórmula:

$$p(\bar{y}) \approx \prod_{i=1}^{|\bar{y}|} \left(\alpha\, p_{NN}(\omega_i|\bar{h}'_i) + (1-\alpha) p_{SRI}(\omega_i|\bar{h}'_i)\right) , \qquad (8.2)$$

donde:

Tabla 8.9: *Topologías de NN LMs utilizados para el modelado de grafemas y tamaños del modelo de lenguaje. //* NN LMs topologies for the character-based task.

	NN LM size			Standard LM size
N-gram	P	H	# of weights	# of N-grams
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

- $\bar{h}'_i = \omega_{i-n+1} \dots \omega_{i-1}$ es el contexto del modelo de N-gramas,
- $p_{SRI}(\omega_i|\bar{h}'_i)$ es la probabilidad del grafema ω_i dado el contexto h'_i, utilizando el modelo de lenguaje de N-gramas estándar estimado mediante SRI,
- $p_{NN}(\omega_i|\bar{h}'_i)$ es la misma probabilidad calculado mediante el N-grama NN LM,
- $\alpha \in [0, 1]$ es el coeficiente de combinación entre el modelo de SRI y el NN LM. Este parámetro se ha optimizado para minimizar la PPL del modelo de lenguaje combinado.

Se ha realizado un barrido de tamaño del valor N del N-grama para ver cómo afecta al resultado del reconocimiento. La tabla 8.9 muestra el tamaño de los modelos, medido como número de pesos para el modelo conexionista, y como número de N-gramas guardados en la tabla para el caso del N-grama de SRI. Para esta experimentación sólo se ha entrenado una única red neuronal por cada valor de N.

La tabla 8.10 y la figura 8.8 muestran la evolución del WER conforme aumentamos el tamaño de la N del N-grama. No se han probado valores de N mayores a 8 debido a problemas computacionales, que podrían ser resueltos implementando una versión de NN LMs especializada para modelos de lenguaje de grafemas. En la tabla 8.10 se pueden consultar los valores del coeficiente de combinación α del NN LM, optimizado para minimizar la PPL. La tabla 8.11 muestra el resultado con la partición de test para el mejor de los modelos encontrados en validación.

8.8.2. Análisis de las palabras fuera de vocabulario

Una ventaja del uso de modelos de lenguaje basados en grafemas es su capacidad de modelar secuencias subléxicas para producir nuevas palabras. El modelo aprende las reglas que subyacen a la formación de palabras a partir de sus grafemas en el lenguaje modelado. A este tipo de aproximaciones se les denomina con léxico libre o "lexicon-free" en inglés, si bien es importante acentuar que el uso de N-gramas con valor de N elevado conlleva que el modelo llegue a recibir como contexto palabras enteras. Esto en cierto modo hace que el modelo aprenda el léxico del conjunto de entrenamiento, si bien es obvio que es mucho más flexible que un modelo de lenguaje basado en palabras.

Tabla 8.10: *WER (%) en el conjunto de validación de la tarea de la IAM-DB para diferentes valores de N usando modelos de lenguaje de grafemas estimados con SRI y con NN LMs. El coeficiente de combinación α se muestra en la última columna. //* WER for the validation set with character LMs.

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

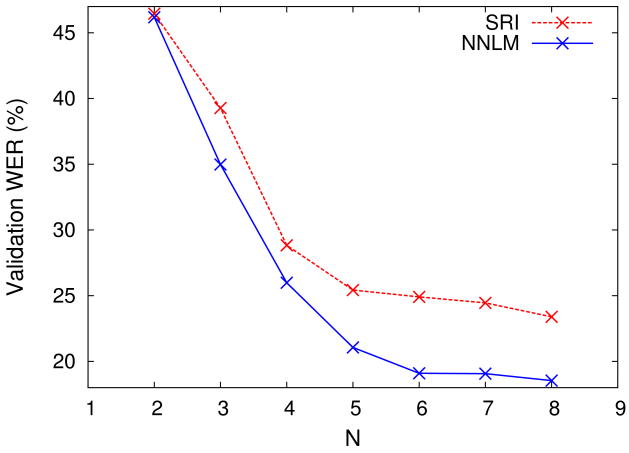


Figura 8.8: *WER y CER (%) en el conjunto de validación de la tarea de la IAM-DB para diferentes modelos de lenguaje de grafemas. //* Plot of the WER and CER for the validation set with character LMs.

Tabla 8.11: *WER y CER en el conjunto de test de la tarea IAM-DB para $N = 8$ y los modelos de lenguaje de grafemas que mejor funcionaron con el conjunto de validación.* // WER and CER for the test set with character LMs.

System	% WER	% CER
SRI	30.9	13.8
NN LM	24.2	10.1
% improvement	21.7	26.8

Tabla 8.12: *Precisión en palabras fuera de vocabulario en el conjunto de test. Existe un total de 544 palabras fuera de vocabulario en dicha partición.* // OOV words accuracy for the test set. There are 554 running OOV words.

System	# OOV words recognized	% accuracy
SRI	162	29.8
NN LM	184	33.8

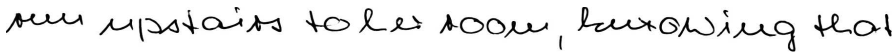
La figura 8.9 muestra algunos ejemplos de transcripciones que contienen palabras que no aparecían en el entrenamiento. Se ha calculado la precisión en el reconocimiento de estas palabras fuera del vocabulario y se muestra en la tabla 8.12. De un total de 544 palabras desconocidas que aparecen en el conjunto de Test, aproximadamente el $34\,\%$ son correctamente reconocidas usando NN LMs de grafemas.

8.8.3. Discusión

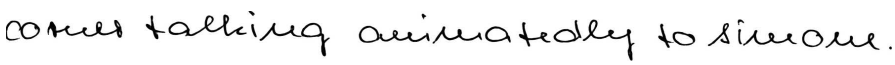
Esta sección ha presentado resultados de reconocimiento utilizando una aproximación al modelado de lenguaje basada en grafemas. Esto es, el sistema busca encontrar la mejor secuencia de grafemas, en lugar de la mejor secuencia de palabras. La segmentación de estos grafemas en palabras se realiza dividiendo la secuencia final por los espacios en blanco, que son modelados como un grafema más, de forma que el reconocedor los puede generar como salida.

Destaca la clara mejora obtenida mediante el uso de modelos conexionistas de grafemas en el proceso de reconocimiento. Se disminuye casi 6.7 puntos absolutos de WER, lo que supone una mejora del $21.7\,\%$ con respecto al modelo de lenguaje basado en N-gramas estándar. No obstante, dado que estamos utilizando modelos a nivel de grafema, el Character Error Rate (CER) es una medida quizá más interesante. En el conjunto de test encontramos que los NN LMs mejoran en 3.7 puntos absolutos de CER a los modelos estimados con SRI. Esto supone una mejora relativa del $26.8\,\%$. Si nos fijamos en la precisión de las palabras desconocidas (véase la tabla 8.12) tenemos que el sistema con NN LMs permite recuperar hasta casi un $34\,\%$ de las palabras desconocidas.

Es posible combinar esta aproximación con un diccionario y un modelo de lenguaje de palabras, lo que nos permitiría mejorar los resultados en las palabras conocidas por el diccionario sin perder la mejora en el reconocimiento de las palabras desconocidas. Podemos

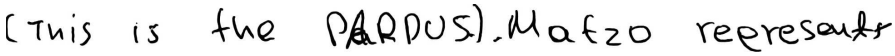


Reference transcription: run upstairs to her room , knowing that
SRI system output: sum upstairs to les today , housekeeping flat
NN LMs system output: sum upstairs toler toour , Louvrewing that



Reference transcripcion: corner talking **animatedly** to Simone .
SRI system output: corcher talking *anilexcitedly* to livedoux .
NN LMs system output: corner talking *animatedly* to liverous .

Reference transcripcion: opinion of **Bassius** that at their desserts the
SRI system output: opinion of *fossils* that at their gressments the
NN LMs system output: opinion of P*assi*ve that at their desserts the



Reference transcripction: (This is the **PARDUS** .) **Matzo** represents
SRI system output: (This is the 1965IO*USI*.*M a* Ezo represents
NN LMs system output: (this is the USSRO*UST-Matzo* represents

Figura 8.9: *Ejemplos de la salida del proceso de decodificación de la tarea IAM-DB con modelo de lenguaje de grafemas. Las palabras fuera de vocabulario se han marcado en negrita. En las transcripciones de los sistemas se han marcado con cursiva los grafemas reconocidos de cada palabra desconocida.* // Instances of some decoding output examples. Bolded words are OOV.

calcular una cota superior de esta combinación. Si existe un $2.86\,\%$ de palabras desconocidas en el conjunto de Test de la IAM-DB (véase la tabla 8.3), y recuperamos el $34\,\%$, dicha cota se aproxima a un punto absoluto de WER ya que las palabras desconocidas son *errores seguros* si se utiliza una aproximación basada en palabras. No obstante, la IAM-DB es una tarea de documentos modernos, y existen numerosos recursos lingüísticos de los que extraer un vocabulario para esta tarea. En tareas donde se trabaja con documentos antiguos o históricos esta mejora puede ser mayor debido a la dificultad de encontrar recursos lingüísticos para esas formas de escritura.

8.9 Experimentos con modelos de palabras y redes neuronales recurrentes

Esta sección describe el trabajo realizado en colaboración con el IAM de la Universidad de Bern y presentado en el congreso ICPR del 2012 [Frinken *et al.*, 2012]. El objetivo de dicho trabajo fue adaptar las ideas de los NN LMs a una red recurrente tipo "Long-Short Term Memory" (LSTM), y que llamaremos LSTM LM. Las LSTMs ya habían sido utilizadas con éxito en diversas tareas de reconocimiento de secuencias [Graves *et al.*, 2008, 2009], siendo este trabajo la primera aproximación de las LSTMs al mundo del modelado de lenguaje. Por sus características concretas, las LSTMs son capaces de modelar dependencias a larga distancia mejor que otro tipo de redes neuronales recurrentes que ya están siendo utilizadas para modelado de lenguaje [Mikolov *et al.*, 2010, 2011]. A pesar de esto, los resultados presentados en esta sección son muy preliminares y no mejoran los obtenidos con NN LMs en la misma tarea.

8.9.1. Modelado de lenguaje con LSTMs

Las redes neuronales recurrentes permiten procesar una secuencia de datos recordando en su memoria la historia de la secuencia que fue procesada con anterioridad. En modelado de lenguaje, esto permite que el término $\bar{h}_j$ del cálculo de la probabilidad condicional $p(y_j|\bar{h}_j)$ no solo sean las dos o tres palabras anteriores, sino, de forma hipotética, todas las palabras anteriores. Dicha longitud en palabras depende de la modelización que el algoritmo de entrenamiento haya encontrado como mejor para la tarea y datos de entrenamiento utilizados. Sin embargo, este tipo de redes neuronales sufren lo que se conoce como *el problema del gradiente desvanecido* (en inglés "vanishing gradient problem"). Tradicionalmente dicho problema ha supuesto una deficiencia en la modelización de las dependencias a larga distancia con redes recurrentes.

En 1997 se propuso una solución a este problema desarrollando un nuevo tipo de neurona con las que describió las redes recurrentes conocidas como LSTM [Hochreiter & Schmidhuber, 1997]. En esta tesis se propuso el uso de las LSTMs tanto para el sistema de reconocimiento como para el modelado de lenguaje. La figura 8.10 ilustra una neurona tipo LSTM, formada por un núcleo que guarda la entrada de la red neuronal con una conexión recurrente que sirve como memoria, y cuatro nodos que se utilizan para controlar el flujo de información hacia dentro y hacia fuera de la neurona:

- Nodo "net input": recibe valores desde la red neuronal y propaga su activación hacia el interior de la neurona.

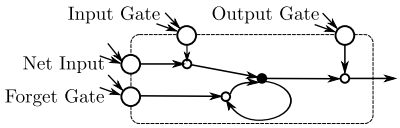


Figura 8.10: *Una neurona tipo LSTM [Frinken* et al.*, 2012].* // An LSTM neuron.

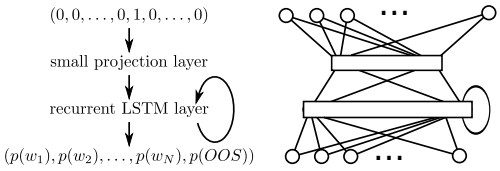


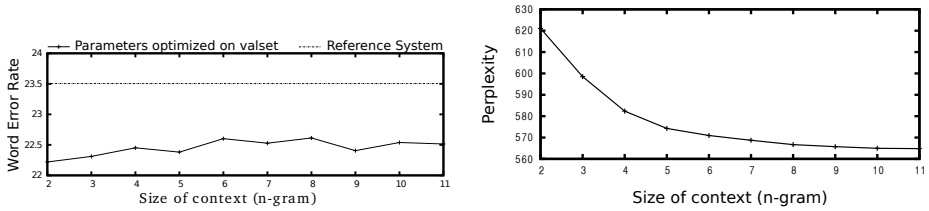
Figura 8.11: *Un modelo de lenguaje tipo LSTM [Frinken* et al.*, 2012].* // An LSTM language model.

- Nodo "input gate": sirve para modular el paso de información al nodo net input. Sólo si el nodo input gate está activo la información llega al núcleo de la neurona.
- Nodo "output gate": de manera similar al anterior pero para controlar la activación de la salida de la neurona. La información contenida en el núcleo de la neurona es propagada hacia fuera sólo si el output gate está activo.
- Nodo "forget gate": cuando este nodo se activa se pone a cero la información contenida en el núcleo de la neurona.

La figura 8.11 ilustra un ejemplo de modelo de lenguaje basado en redes LSTM. Se utilizan las mismas ideas que para los NN LMs, siendo la entrada de la red neuronal una proyección de las palabras en un espacio continuo. Utilizaremos la idea de la Short-List para simplificar el entrenamiento. Tanto la entrada como la salida de la red neuronal utilizarán dicha Short-List, compuesta por aquellas palabras que aparecen más de $\Theta = 21$ veces en el conjunto de entrenamiento. La masa de probabilidad de las palabras fuera de la Short-List (OOS) se calcula mediante la ecuación (3.5).

La red neuronal LSTM LM está formada por:

- Una entrada codificada localmente donde se reciben las $N-1$ palabras anteriores. Esto permite que en el cálculo de la probabilidad condicional se utilice el contexto de las $N-1$ palabras anteriores además de la memoria de la LSTM.
- Una capa de proyección que reduce la dimensión de la entrada y codifica las palabras de entrada en un espacio continuo. Esta capa de proyección tendrá un tamaño de 192 neuronas. Para simplificar el entrenamiento, se utilizará la capa de proyección resultado de entrenar los NN LMs de la sección 8.7.1.
- Una capa de neuronas tipo LSTM con recurrencia, formada por 100 neuronas.
- Una capa de salida con función de activación softmax que permite computar la probabilidad de todas las palabras del Short-List, incluida la neurona extra que sirve para representar al conjunto de todas las palabras fuera de la Short-List.



(a) Recognition accuracy. (b) Perplexity.

Figura 8.12: *Gráfica del % WER y la PPL con el conjunto de test del corpus IAM-DB y diferentes valores del orden N [Frinken* et al.*, 2012].* // PPL and WER for the test set.

8.9.2. Sistema de reconocimiento basado en LSTMs

La ventaja de utilizar LSTM LMs es que pueden considerar el contexto completo de la frase para calcular la probabilidad condicional de la siguiente palabra. Sin embargo, esto conlleva una explosión exponencial del coste del algoritmo, ya que el espacio de búsqueda pasa de ser un grafo a convertirse en un árbol de prefijos. Para reducir este coste, las LSTM LMs serán utilizadas en una etapa de rescoring de listas de N-best. Dichas listas serán generadas por un sistema de reconocimiento como el utilizado en [Graves *et al.*, 2009], formado por un reconocedor basado en una red LSTM (en sustitución de los HMMs) y un modelo de lenguaje de bigramas (el modelo de lenguaje **mKN** descrito en la sección 8.7.1, formado por el vocabulario de 103K palabras). La descripción de los modelos ópticos basados en LSTMs se puede consultar en [Graves *et al.*, 2009].

8.9.3. Experimentación

El entrenamiento del LSTM LM se ha realizado utilizando el corpus descrito en la sección 8.5. El modelo final es una combinación lineal de cuatro redes neuronales, optimizada para reducir la perplejidad en el conjunto de validación del corpus IAM-DB.

Para llevar a cabo los experimentos, primero el reconocedor basado en LSTMs decodifica cada línea de texto utilizando un modelo de lenguaje estándar de bigramas. El resultado de cada decodificación es una lista de las $5\,000$-best hipótesis. Sobre esa lista se hace un rescoring utilizando el LSTM LM, combinado con la probabilidad del bigrama original. El GSF y el WIP se optimizan para minimizar el WER en el conjunto de validación del corpus IAM-DB.

Una red recurrente calcula la probabilidad condicional de la siguiente palabra dada la historia de todas las anteriores dándole como entrada únicamente una palabra cada vez. Sin embargo, es posible que el modelo pueda estimar mejor dicha distribución de probabilidad si en lugar de darle una palabra se utiliza un contexto de las $N-1$ palabras anteriores, siendo N el orden del N-grama. Para validar esta hipótesis se han probado diferentes combinaciones de LSTMs, modificando el orden N del modelo. La figura 8.12(a) ilustra los resultados obtenidos. Se puede observar que el sistema baseline con bigrama estándar obtiene un WER de $23.5\,\%$, y el LSTM LM logra resultados que están entre el $22.2\,\%$ y el $22.4\,\%$, mejorando de forma sistemática el baseline.

Adicionalmente, la figura 8.12(b) ilustra la PPL en el conjunto de test del LSTM LM en función del orden N. Se puede observar una correlación muy clara entre N y la PPL resultante.

8.9.4. Conclusiones

Esta sección ha propuesto el uso de redes recurrentes tipo LSTM para el modelado de lenguaje. El diseño de esta arquitectura de redes recurrentes para procesar dependencias a larga distancia promete mejorar los resultados en modelado del lenguaje.

En los experimentos realizados, los LSTM LMs han superado en todos los casos el baseline utilizado. Se ha observado una dependencia clara del tamaño del contexto de entrada de la LSTM LM con la PPL del modelo, si bien en WER no se ha observado dicha correlación. Este comportamiento puede ser debido a las características de la tarea escogida para evaluar los modelos, que no parece ser la más adecuada, puesto que la IAM-DB tiene una media de 9 palabras por línea, lo que reduce significativamente las ventajas de las LSTMs en el procesamiento de dependencias a larga distancia. Por tanto, en dicha tarea un NN LM estándar es suficiente para mejorar los resultados.

La experimentación realizada demuestra la viabilidad de utilizar LSTM LMs, aunque es necesario profundizar más en su investigación. Una posible solución al problema que se ha encontrado en el reconocimiento de líneas es concatenar todas las líneas que forman una frase, o bien procesar un párrafo completo cada vez. Esto permitiría al LSTM LM guardar en su memoria la información completa de dicha frase (o párrafo) y sacar provecho de las dependencias a larga distancia entre las palabras que este modelo de lenguaje es capaz de modelar.

8.10 Resumen

Este capítulo ha descrito los modelos ópticos utilizados para la experimentación en reconocimiento de escritura manuscrita de esta tesis, descritos en [España-Boquera *et al.*, 2011], si bien dicha publicación y trabajo no forman parte central de esta tesis.

Se ha presentado la aplicación de NN LMs a tareas de reconocimiento automático de escritura manuscrita, estudiándose diferentes tamaños de vocabulario a la entrada del NN LM, así como la comparación entre la aproximación integrada durante la etapa de decodificación frente a la aproximación basada en el rescoring de listas de N-best.

Se ha estudiado el efecto que tiene utilizar modelos basados en palabras frente a modelos basados en grafemas, encontrando que, si bien la aproximación basada en palabras obtiene mejores resultados, la aproximación basada en grafemas logra resultados competitivos. Esto sugiere que para tareas donde el vocabulario no esté acotado, la aproximación basada en grafemas puede ser más adecuada.

También se ha estudiado de forma preliminar la capacidad de las redes recurrentes tipo LSTM para modelado de lenguaje. Los resultados obtenidos sugieren una investigación en tareas donde se pueda obtener más provecho a dichos modelos.

En la experimentación de este capítulo se observa una clara y robusta mejora de los NN LMs frente a los modelos de lenguaje estándar. Estos resultados están en consonancia con los obtenidos por NN LMs en otras tareas.

Recapitulando, en cuanto a objetivos científicos este capítulo ha logrado:

- Contrastar los resultados obtenidos por diferentes configuraciones de NN LMs frente a modelos de lenguaje estándar para tareas de HTR. Los resultados presentados son los mejores conocidos y estamos preparando un envío al "IEEE Transactions on Neural Networks". Se ha obtenido un WER de 20.2 puntos y un CER de 8.3 puntos con el mejor de los sistemas que usa NN LMs.

- En colaboración con otros autores, hemos evaluado los NN LMs en otros sistemas de reconocimiento de escritura, basados en redes recurrentes BLSTM, haciendo un rescoring de las N-best hipótesis de salida del sistema. En este caso, también se mejoran los resultados. Adicionalmente, se ha combinado nuestro sistema y el basado en BLSTM, usando la aproximación "Recognizer Output Voting Error Reduction" (ROVER) [Fiscus, 1997], mejorando aún más las prestaciones. Este trabajo se ha enviado a la revista "Pattern Recognition" [Zamora-Martínez *et al.*, 2012c] y recoge algunos de los resultados presentados en este capítulo.

- Medir la capacidad de un reconocedor basado en modelos de lenguaje de grafemas, y contrastar la ventaja de usar NN LMs frente a no hacerlo. Estos resultados han sido presentados en [Zamora-Martínez *et al.*, 2010]. El error obtenido por el sistema basado en NN LMs ha sido de 10.1 puntos de CER, una mejora relativa del $26.8\,\%$ frente al sistema que utiliza modelos de lenguaje estándar (N-gramas estimados con SRI).

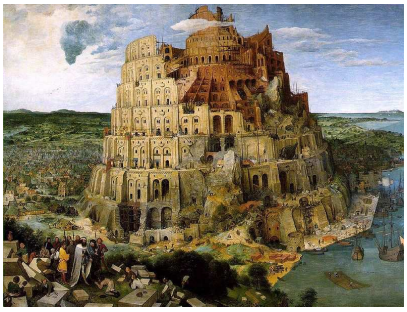
- Presentar el uso de redes recurrentes tipo LSTM para modelar el lenguaje, obteniendo resultados muy prometedores que sugieren un cambio de tarea para sacar el máximo partido a las características más importantes de las LSTMs. Este trabajo ha sido realizado en colaboración con el IAM de la Universidad de Bern y los resultados han sido publicados en [Frinken *et al.*, 2012].

Parte III

Aplicación de NN LMs a la traducción automática estadística

9

TRADUCCIÓN AUTOMÁTICA ESTADÍSTICA BASADA EN SEGMENTOS Y N-GRAMAS



Índice

Este capítulo detalla las bases formales de la traducción automática estadística (SMT). Aunque la traducción automática es un proceso verdaderamente complejo, únicamente bien realizado por especialistas, en la literatura se pueden encontrar diversas formas y algoritmos que tratan de solucionar el problema de la traducción de forma automatizada. Este capítulo comienza estableciendo de forma sencilla las bases de la traducción automática pre-estadística, para más tarde describir la traducción basada en segmentos. En ese momento daremos el salto a los transductores de estados finitos, y la traducción basada en modelos de lenguaje N-gramas.

9.1 Introducción

La traducción automática trata el problema de convertir una secuencia de palabras en una lengua determinada (origen) en una secuencia de palabras distinta en otra lengua (destino), de tal forma que una sea traducción de la otra. También es posible reformular el problema para traducir de forma directa a partir una elocución de voz de una persona [Vidal, 1997], e incluso es posible transformar la hipótesis textual encontrada por el sistema en una elocución sintetizada. Cualquiera que sea el problema para el cual necesitamos un sistema traducción automática, existen diversas aproximaciones en la literatura. Comenzaremos hablando de las aproximaciones clásicas a la traducción automática, para más tarde presentar la aproximación estadística en la que se basa este trabajo.

9.1.1. Traducción automática clásica

En la aproximación clásica el problema de la traducción automática se clasifica en función de la capa de abstracción en la que se implementa el proceso de traducción o *transferencia*. En función de si este proceso se ejecuta sobre la capa morfológica (a nivel de palabras), sobre la capa sintáctica, semántica o conceptual, podemos encontrar los siguientes tipos de sistemas de traducción (véase también la figura 9.1):

- Traducción directa: se procede palabra-a-palabra sobre el texto origen, y mediante un enorme diccionario bilingüe se genera la mejor traducción de cada palabra.
- Transferencia sintáctica: primero se extrae la estructura sintáctica del texto origen. Después se aplican reglas que permiten transformarla en la estructura sintáctica del texto escrito en la lengua destino. Finalmente se genera el texto de salida a partir de la estructura sintáctica.
- Transferencia semántica: se extrae la estructura semántica del texto origen. Después se aplican reglas que permiten transformarla en la estructura semántica del texto escrito en la lengua destino. Finalmente se genera el texto de salida a partir de dicha estructura semántica.
- Interlingua: el texto origen se transforma a algún lenguaje semántico intermedio, al que llamamos interlingua. A partir de esta representación se genera el texto en la lengua destino.

Esta forma de entender la traducción automática inicia los pasos más importantes a seguir en función de la representación que estemos usando.

9.1.2. Traducción automática estadística

La traducción automática clásica se focaliza en el proceso de traducción en sí mismo, dando la descripción de pasos a seguir para realizar la traducción. En la traducción automática estadística hay un cambio de paradigma. El proceso pierde importancia, siendo el resultado de la traducción el foco sobre el que giran los algoritmos. Bajo este paradigma es necesario cuantificar de algún modo cuando una frase es una traducción fidedigna y cuando es fluida. Una traducción puede ajustarse mucho a la verdad de lo que dice la frase original, pero no ser

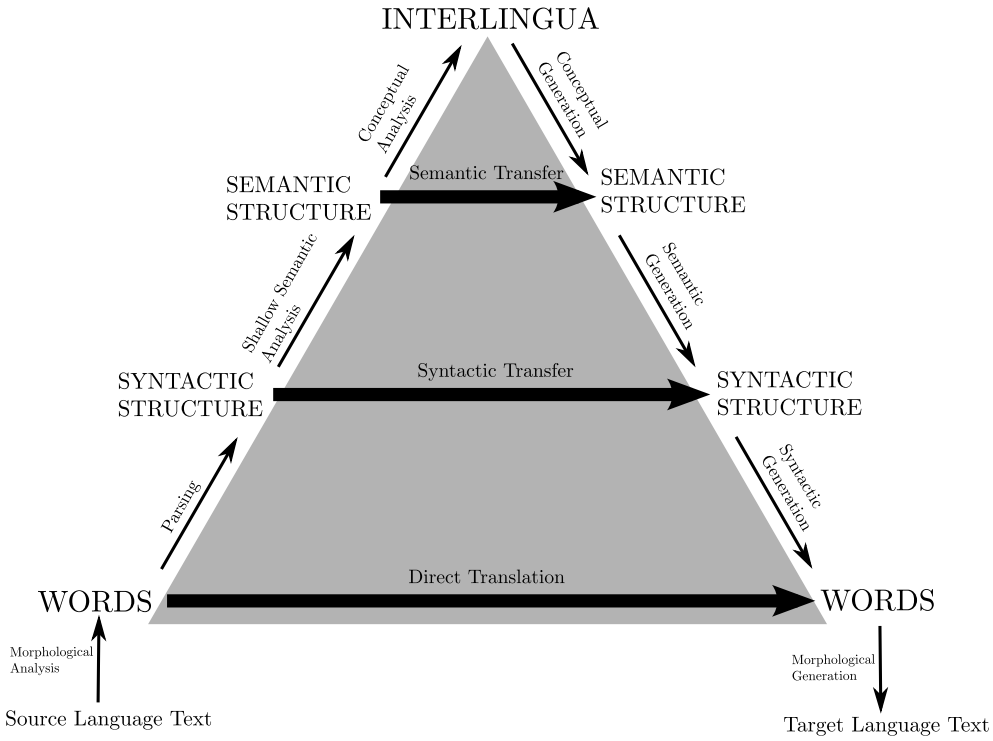


Figura 9.1: *Triángulo de Vauquois, especifica los diferentes niveles de abstracción definidos por la traducción automática clásica [Jurafsky & Martin, 2009].* // Vauquois triangle.

fluida en la lengua destino. Por otro lado una traducción puede ser muy fluida en la lengua destino pero no ajustarse a la realidad que debería expresar.

En traducción automática estadística el proceso de traducción se modela como la producción de una salida que maximiza el valor de ambas funciones objetivo. Desde este punto de vista se entiende la traducción como:

$$\text{mejor traducción } \hat{\bar{y}} = \arg\max_{\bar{y}} \text{fidelidad}(\bar{y},\bar{x})\ \text{fluidez}(\bar{y})$$

que se corresponde claramente con la ecuación (1.2), la ecuación básica del reconocimiento de formas. Más formalmente:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{y}|\bar{x}) = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y}) = \quad (1.2)$$

$$= \arg\max_{\bar{y}\in\Omega^+} p(\bar{x},\bar{y}) \quad (9.1)$$

donde:

- Ω es el vocabulario de la lengua destino para tarea;
- $\bar{x} = x_1 \dots x_{|\bar{x}|} \in \Sigma^+$ es la frase de entrada en la lengua origen, compuesta por palabras del vocabulario Σ;
- $\bar{y} = y_1 \dots y_{|\bar{y}|} \in \Omega^+$ es la frase generada como salida del sistema en la lengua destino.

Estas ecuaciones indican que el cálculo de la probabilidad de la traducción se puede realizar dentro del marco estadístico de dos maneras:

- a partir de la combinación de la probabilidad de la traducción inversa $p(\bar{x}|\bar{y})$ y del modelo de lenguaje $p(\bar{y})$ (ecuación (1.2));
- a través de la probabilidad conjunta $p(\bar{x},\bar{y})$ (ecuación (9.1)).

En ambos casos estamos produciendo una salida que maximiza tanto la corrección de la traducción como su fluidez.

9.2 Modelo de traducción basado en segmentos

El objetivo del modelo de traducción es asociar la probabilidad con la que la frase origen $\bar{x}$ genera la frase destino $\bar{y}$. Actualmente, la aproximación más utilizada consiste en calcular esa probabilidad considerando segmentos de ambos pares de frases, donde cada segmento es una secuencia más o menos larga de palabras. Estas secuencias habrán sido vistas en un corpus de entrenamiento y se habrán estimado las probabilidades de la traducción entre segmentos mediante técnicas de conteo que describiremos más adelante (sección 9.5).

Los modelos estadísticos de traducción se basan en la idea del alineamiento de palabras. Un alineamiento de palabras es una asignación entre las palabras de la frase origen y las palabras de la frase destino, como indica el ejemplo de la figura 9.2.

En un principio podemos encontrar relaciones arbitrarias de alineamiento entre palabras. No obstante, en la práctica se usan los modelos de IBM [Brown *et al.*, 1990] que imponen

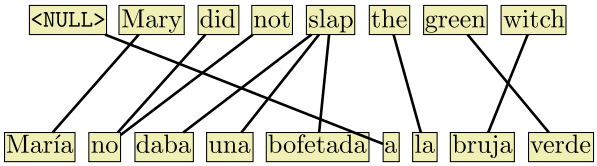


Figura 9.2: *Alineamiento de las palabras entre una frase en inglés y su traducción al castellano. <NULL> hace referencia a la palabra vacía. //* Instance of the words alignment between English sentence and its translation to Spanish. Referenced at page 310.

restricciones a la forma de dicha relación de alineamiento. La adición de la palabra especial <NULL> permitirá modelar la inserción y borrado de palabras. No vamos a entrar en el detalle de los modelos de alineamiento, pues no son parte importante en el foco de esta tesis. A partir de dicho alineamiento se calculan tres cosas:

- diccionario estadístico de traducción palabra-a-palabra;
- unidades básicas de la traducción (segmentos, tuplas bilingües);
- distribución de probabilidad de los modelos de traducción sobre los segmentos.

9.2.1. Simetrización de los alineamientos para SMT basada en segmentos

El alineamiento a nivel de palabras genera información en dos direcciones:

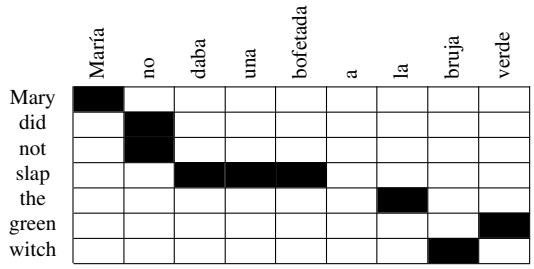
- directa con la traducción: indica la forma en que se alinean las palabras de la frase origen respecto a la frase destino;
- inversa con la traducción: indica la forma en que se alinean las palabras de la frase destino respecto a la frase origen.

Estas dos direcciones del alineamiento no son simétricas. Se han propuestos muchos heurísticos [Och & Ney, 2003; Koehn *et al.*, 2003] que permiten simetrizar estas dos direcciones para obtener un único alineamiento que al final agrupe a ambas y del cual podamos extraer los segmentos del modelo de traducción y hacer las cuentas.

9.2.2. Extracción de pares de segmentos alineados

Una vez tenemos las dos direcciones del alineamiento combinadas, se tiene una única forma de alinear las palabras de cada par de frases. La figura 9.3 muestra la matriz de alineamiento para la frase del ejemplo 9.2 y algunos ejemplos de pares de segmentos que se pueden extraer de dicho alineamiento.

El procedimiento de extracción de segmentos alineados obtiene, a partir del alineamiento entre palabras, *todos* los pares de segmentos *posibles* que se pueden generar cumpliendo el criterio de que sean consistentes. Un par de segmentos es *consistente* si todas las palabras que forman parte del par de segmentos se alinean entre sí y *nunca* con una palabra de fuera del par.



(María, Mary), (no, did not),

(slap, daba una bofetada), (verde, green),

(a la, the), (bruja, witch),

(María no, Mary did not),

(no daba una bofetada, did not slap),

(daba una bofetada a la, slap the),

(bruja verde, green witch),

(a la bruja verde, the green witch), ...

Figura 9.3: *Un ejemplo de alineamiento aplicando los heurísticos de intersección y añadiendo puntos extraídos de la unión. A la derecha, algunos pares de segmentos consistentes con el alineamiento. //* Instance of word alignment extracted from intersection+union heuristic (left). Some consistent phrase pairs (right). Referenced at page 310.

9.2.3. Combinación log-lineal de modelos

Los sistemas de traducción actuales, como ya hemos comentado en este texto, han dejado de lado la elegancia matemática del marco estadístico más estricto. Los mejores resultados en SMT se obtienen a través del marco de la máxima entropía, donde se establece la probabilidad de la traducción como una combinación log-lineal de una serie de M modelos diferentes, donde cada uno de ellos produce una puntuación (entendida como probabilidad para muchos de estos modelos). La puntuación final es una combinación de la puntuación de cada uno de los M modelos. A cada modelo se le asigna un coeficiente de mezcla en dicha combinación [Och & Ney, 2002] siguiendo la ecuación (1.3), que retomamos aquí:

$$\hat{\bar{y}} = \arg\max_{\bar{y}} \prod_{m=1}^{M} \mathcal{H}_m(\bar{x}, \bar{y})^{\lambda_m} \qquad (1.3)$$

donde λ_m es el coeficiente de mezcla del modelo m y $\mathcal{H}_m(\bar{x}, \bar{y})$ es la puntuación (o probabilidad) asociada por el modelo m a la traducción, y M el número de modelos disponibles.

9.3 Aproximación basada en N-gramas: transductores de estados finitos

La aproximación seguida en este trabajo se basa en el trabajo de [Mariño *et al.*, 2006], que tiene muchos detalles en común con la traducción basada en segmentos. La mayor diferencia se encuentra en la forma en que son extraídos los pares de segmentos alineados. Bajo esta aproximación, dado un par alineado de frases en las dos lenguas, se establece una única secuencia de pares de segmentos alineados que minimiza el número de palabras que forman parte de cada par. A cada una de las unidades de traducción en las que se segmenta cada par de frases la llamaremos *tupla bilingüe*. El entrenamiento del modelo de traducción se

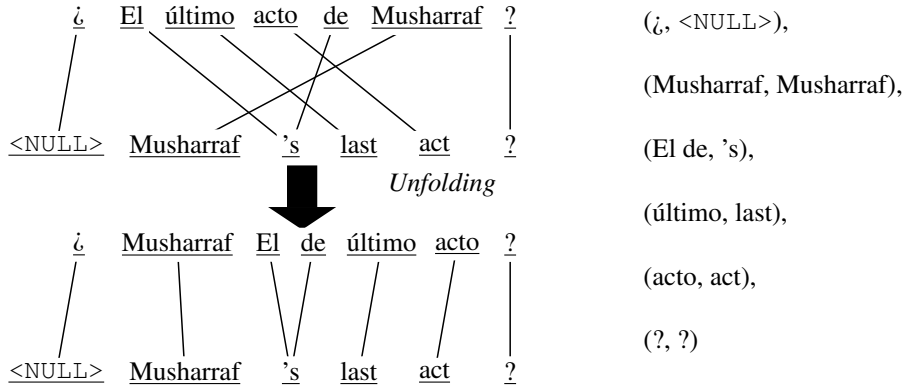


Figura 9.4: *Ejemplo de alineamiento entre dos frases y el conjunto de tuplas extraídas. <NULL> representa la palabra vacía.* // Instance of a word aligment and the set of bilingual tuples extracted from it. Referenced at page 311 .

sigue de aplicar la metodología "Grammar Inference and Alignments for Transducer Inference" (GIATI) [Casacuberta & Vidal, 2004], que permite, dados los alfabetos origen y destino, generar un alfabeto extendido que será el *alfabeto de tuplas*.

Por lo tanto, cada tupla está compuesta por una secuencia de palabras en la lengua origen, y su correspondiente traducción en la lengua destino. Estas son extraídas de un corpus bilingüe alineado palabra a palabra entre ambas lenguas. Dicho alineamiento se calcula mediante la herramienta GIZA++ [Och & Ney, 2003] que implementa diversos modelos IBM de alineamiento y traducción [Brown *et al.*, 1993].

Las tuplas se extraen asumiendo alineamientos *muchos a muchos*, en contra de otras aproximaciones similares que asumen *uno a uno* [Bangalore & Riccardi, 2000], o bien *uno a muchos* [Casacuberta & Vidal, 2004].

Otra diferencia con la traducción basada en segmentos es la monotonización del orden de las palabras de la frase origen. El reordenamiento de la frase de entrada [Banchs *et al.*, 2005; Sanchis & Casacuberta, 2006; Costa-jussà & Fonollosa, 2006], de tal forma que las palabras sigan el orden de sus *homólogas* en la frase destino, aumenta la calidad y *reusabilidad* de las tuplas extraídas. Este proceso hace que la frase de entrada tenga el orden adecuado para la dirección de la traducción que se está trabajando. Denominaremos a este procedimiento "unfolding".

El proceso de extracción de tuplas es el descrito en [Banchs *et al.*, 2005]. La figura 9.4 muestra un ejemplo de este proceso a partir de dos frases alineadas. El procedimiento se describe más en detalle en la sección 9.4.1. Tendremos en cuenta las consideraciones explicadas en [Banchs *et al.*, 2005; Mariño *et al.*, 2006] para implementar la búsqueda con reordenamiento de la frase de entrada.

Los modelos de traducción basados en tuplas bilingües y los modelos de traducción basados en segmentos son en esencia muy similares, aunque están inspirados en ideas diferentes. Los primeros se inspiran en la traducción basada en transductores estocásticos de estados finitos [Vidal *et al.*, 2005], mientras que los segundos se basan en una generalización del concepto de palabra, y de los diccionarios de traducción palabra a palabra. A pesar de estas diferencias, esencialmente ambos modelos pueden reducirse a un mismo formalismo. Los

modelos basados en transductores estocásticos son más generales, e incluyen a los modelos basados en segmentos. Bastaría con representar un autómata que transita a un estado único con todos los pares de segmentos en bucles sobre dicho estado. Se transita sobre dicho autómata con las palabras del origen, y se emite o bien el par de segmentos, o bien las palabras de la lengua destino.

9.4 Estimando N-gramas de tuplas bilingües

Los transductores estocásticos de estados finitos (SFSTs de sus siglas en inglés) son una extensión de los autómatas estocásticos de estados finitos [Vidal *et al.*, 2005] utilizando dos alfabetos en lugar de uno, el alfabeto de entrada Σ (el vocabulario de la lengua origen), y el alfabeto de salida Ω (el vocabulario de la lengua destino). En la definición original las transiciones del transductor están etiquetadas con una palabra de entrada y una secuencia de palabras de salida, pudiendo cualquiera de ellas ser de longitud 0 (equivalente a transitar con la palabra vacía o nula).

Adaptaremos esta descripción a nuestra modelización, de manera que las transiciones contienen una secuencia de palabras de Σ de longitud mayor que 0, y una secuencia de palabras de Ω de longitud mayor o igual 0. Formalmente representaremos nuestra adaptación como $\mathcal{ST} = \langle \Sigma, \Omega, \mathcal{Q}, q_0, P_T, P_F \rangle$, donde:

- $\mathcal{Q} = \{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$ es el conjunto de estados del transductor,
- $q_0 \notin \mathcal{Q}$ es el estado inicial,
- $P_T : (\mathcal{Q} \cup q_0) \times \Sigma^+ \times \Omega^* \times \mathcal{Q} \rightarrow [0, 1]$ es la función de transición estocástica que define la probabilidad de ir de un determinado estado a otro consumiendo una determinada secuencia de palabras y emitiendo una secuencia determinada de palabras,
- $P_F : \mathcal{Q} \rightarrow [0, 1]$ es la probabilidad de ser final un determinado estado.

El transductor así definido deberá cumplir, $\forall q_i \in (\mathcal{Q} \cup q_0)$, la siguiente condición para que sea estocástico:

$$\left(P_F(q_i) + \sum_{(x', y', q') \in \Sigma^+ \times \Omega^* \times \mathcal{Q}} P_T(q_i, x', y', q') \right) = 1 \,.$$

Estimamos un SFST a partir de un corpus bilingüe alineado a nivel de frase, y dentro de cada frase a nivel de palabra, utilizando la técnica GIATI [Casacuberta & Vidal, 2004]. Esta se basa en el siguiente teorema:

Teorema 2 *Todo transductor estocástico de estados finitos puede ser aprendido a partir de un lenguaje regular estocástico y dos morfismos.*

Definiremos un nuevo alfabeto Δ compuesto por el conjunto de posibles $z \in \Sigma^+ \times \Omega^*$ tales que $\exists_{q_i, q_j} P_T(q_i, x', y', q_j) > 0$, siendo $z = (x', y')$. A cada una de las posibles z las llamaremos *tupla* o *tupla bilingüe*. Este alfabeto *extendido* a partir de los dos originales nos permitirá redefinir el transductor estocástico extendido, $\mathcal{ST}' = \langle \Delta, \mathcal{Q}, q_0, P_T, P_F \rangle$, donde:

- Δ es el alfabeto extendido que contiene a todas las posibles tuplas del transductor $\mathcal{ST}$ original,
- $\mathcal{Q}$ es el conjunto de estados, idéntico al original,
- q_0 es el estado inicial,
- $P_T : \mathcal{Q} \times \Delta \times \mathcal{Q} \rightarrow \mathbb{R}^{>0}$ es la función estocástica de transición,
- $P_F : \mathcal{Q} \rightarrow [0, 1]$ es la probabilidad final de cada estado.

Y definiremos los siguientes morfismos para llevar a cabo la transformación, dos necesarios para aplicar GIATI, y uno más que nos permitirá simplificar la notación:

- $\mathcal{L} : \Sigma^+ \times \Omega^* \rightarrow \Delta$, aplicación biyectiva que permite, dado un segmento en la lengua origen y su correspondiente traducción en la lengua destino, obtener la correspondiente tupla bilingüe del alfabeto extendido Δ,
- $t : \Delta \rightarrow \Omega^*$, aplicado sobre una tupla, devuelve la parte de la misma que se corresponde con el segmento de la lengua destino,
- $s : \Delta \rightarrow \Sigma^+$, aplicado sobre una tupla, devuelve la parte correspondiente al segmento de la lengua origen.

Para producir esta transformación hemos seguido los pasos básicos de GIATI. Dado D, un conjunto de pares de entrenamiento $(\bar{x}, \bar{y}) \in \Sigma^+ \times \Omega^+$, se siguen estos pasos:

1. Cada par de entrenamiento $(\bar{x}, \bar{y}) \in D$ se transforma en una secuencia $\bar{\mathcal{T}} \in \Delta^+$ del alfabeto extendido Δ, generando un nuevo conjunto D'.
2. A partir de D' se estima un autómata finito estocástico mediante alguna técnica de inferencia, en este caso mediante N-gramas.
3. Los símbolos de las transiciones del autómata (que pertenecen al alfabeto Δ) se transforman en los símbolos de entrada/salida del autómata ($\Sigma^+ \times \Omega^*$).

El modelo $\mathcal{ST}'$ así estimado, dado un par alineado de frases (x, y) en ambas lenguas y segmentado en tuplas, aproxima la probabilidad conjunta $p(x, y)$ y podría servir como modelo de traducción en un sistema estadístico clásico, a través de la ecuación (9.1). Sin embargo, estamos trabajando dentro del marco de máxima entropía, por lo que combinaremos este modelo de traducción con la información aportada por otras fuentes de conocimiento (modelos $\mathcal{H}_m$ de la ecuación (1.3)).

9.4.1. Segmentando el corpus bilingüe

Basándonos en todas las definiciones anteriores, vamos a estimar el modelo de lenguaje bilingüe para el transductor $\mathcal{ST}'$, que tendrá como alfabeto a Δ. Adaptando el trabajo de [Llorens Piñana, 2000], es posible convertir un modelo de lenguaje en un autómata de estados finitos. Si estimamos dicho modelo a partir de un corpus bilingüe alineado y segmentado en unidades del alfabeto Δ, tendremos automáticamente el correspondiente autómata $\mathcal{ST}'$.

Para segmentar el corpus partimos del alineamiento a nivel de palabras calculado mediante GIZA++ [Och & Ney, 2003], y la técnica del unfolding. Dicha segmentación sigue los siguientes pasos:

1. Reordenamos las palabras de la frase origen, para que den cuenta del alineamiento obtenido con GIZA++ (unfolding).

2. Formaremos segmentos de palabras lo más pequeños posible en ambas lenguas, de tal forma que todas las palabras del grupos estén alineadas con palabras de un mismo grupo en la otra lengua (criterio de *consistencia* en modelos basados en segmentos).

3. Una vez extraídas las tuplas, volvemos a ordenar la parte origen de la tupla para que de cuenta del ordenamiento original de la frase. Esto se hace así para que el alineamiento dentro de una tupla sea monótonamente creciente, simplificando la búsqueda.

Si nos fijamos con detalle en estas reglas, aparecen tuplas que en la parte origen tienen discontinuidades producidas por el primero de los pasos (el unfolding). Esto habrá que tenerlo en cuenta ya que será importante de cara a definir el algoritmo de búsqueda y los modelos de distorsión. Véase la sección 9.5.5 para entender mejor esta problemática.

Este es el modelo de traducción básico de la aproximación elegida. A dicho modelo de traducción se le añaden otros modelos, como son:

- las probabilidades léxicas basadas en modelos IBM-1 asociadas a cada tupla,
- probabilidad de traducción directa e inversa asociada a cada tupla,
- penalización del número de palabras y de tuplas generadas por el sistema,
- y un modelo de lenguaje de la lengua destino.

9.5 Modelos utilizados en la combinación

El sistema presentado implementa la búsqueda de la mejor traducción dentro de una combinación log-lineal de diversos modelos. Los modelos de la combinación log-lineal computan su puntuación a partir de la secuencia de tuplas producida como hipótesis, si bien esta computación se descompone aplicando la regla de la cadena para poder hacer los cálculos durante el procedimiento de decodificación. Adaptaremos la ecuación (1.3) para el caso de una hipótesis segmentada en tuplas de esta manera:

$$\hat{\bar{\mathcal{T}}} = \arg\max_{\bar{\mathcal{T}}} \prod_{m=1}^{M} \mathcal{H}_m(\bar{\mathcal{T}})^{\lambda_m} . \tag{9.2}$$

En lo que sigue denotaremos a una secuencia de tuplas como $\bar{\mathcal{T}} \in \Delta^*$, a una única tupla como $z \in \Delta$, un segmento de palabras de la lengua origen como $x \in \Sigma^*$, y un segmento de palabras de la lengua destino $y \in \Omega^*$. Dada la secuencia $\bar{\mathcal{T}}$, la secuencia de palabras correspondiente en la lengua origen será $\bar{x}$ y la correspondiente en la lengua destino $\bar{y}$.

Esta ecuación necesita ser extendida para que la búsqueda se realice sobre todos los posibles reordenamientos. En ese caso el algoritmo busca de forma conjunta la mejor secuencia de tuplas y el mejor reordenamiento de las palabras de entrada $\bar{x}$:

$$(\hat{\bar{\mathcal{T}}}, \hat{\varphi}) = \arg\max_{(\bar{\mathcal{T}},\varphi)} \prod_{m=1}^{M} \mathcal{H}_m(\bar{\mathcal{T}}, \varphi)^{\lambda_m} , \tag{9.3}$$

donde:

$$\varphi : \{1, 2, \ldots, |\bar{x}|\} \rightarrow \{1, 2, \ldots, |\bar{\mathcal{T}}|\}\,,$$

es una función que asocia un índice de tupla a cada posición de la frase de entrada $\bar{x}$. El modelo *obliga* a que el orden de las posiciones de las palabras dentro de una tupla sea siempre creciente.

9.5.1. Modelos incontextuales

Definiremos como modelos incontextuales a todos aquellos que para una tupla $z \in \Delta$ siempre ofrecen el mismo resultado, sin tener en cuenta el contexto de las tuplas que hay alrededor de z en la hipótesis. Estos modelos se calculan a priori y se almacena su puntuación en la tabla de tuplas (o tabla de segmentos).

Modelos léxicos basados en IBM-1

Permiten establecer una penalización a priori de cada una de las tuplas [Mariño *et al.*, 2006]. Para cada tupla del sistema de traducción se precalcula su puntuación y se guarda en una tabla. Está basado en las probabilidades de los modelos IBM-1 [Brown *et al.*, 1993]:

$$\mathcal{H}_{s2t}(\bar{\mathcal{T}}, \varphi) = \sum_{i=1}^{|\bar{\mathcal{T}}|} \mathcal{H}'_{s2t}(s(\mathcal{T}_i), t(\mathcal{T}_i))\,, \tag{9.4}$$

$$\mathcal{H}_{t2s}(\bar{\mathcal{T}}, \varphi) = \sum_{i=1}^{|\bar{\mathcal{T}}|} \mathcal{H}'_{t2s}(s(\mathcal{T}_i), t(\mathcal{T}_i))\,, \tag{9.5}$$

donde $\mathcal{H}'_{s2t}$ y $\mathcal{H}'_{t2s}$ se definen como:

$$\mathcal{H}'_{s2t}(x, y) = \frac{1}{(|x|+1)^{|y|}} \prod_{j=1}^{|y|} \sum_{i=0}^{|x|} q(y_j|x_i)\,, \tag{9.6}$$

$$\mathcal{H}'_{t2s}(x, y) = \frac{1}{(|y|+1)^{|x|}} \prod_{i=1}^{|x|} \sum_{j=0}^{|y|} q(x_i|y_j)\,, \tag{9.7}$$

siendo $q(y_j|x_i)$ es la probabilidad de traducción de la palabra x_i en la palabra y_j, y $q(x_i|y_j)$ la inversa (diccionario léxico). Estas probabilidades se computan por conteo del número de veces que ambas palabras han sido alineadas, a partir del corpus bilingüe alineado a nivel de palabras:

$$q(y|x) = \frac{\mathcal{C}(x, y)}{\sum_{y'} \mathcal{C}(x, y')}\,, \tag{9.8}$$

$$q(x|y) = \frac{\mathcal{C}(x, y)}{\sum_{x'} \mathcal{C}(x', y)}\,. \tag{9.9}$$

$$\tag{9.10}$$

Probabilidad condicional de la traducción directa e inversa

Se estiman por conteo en el corpus bilingüe:

$$\mathcal{H}_{p(y|x)}(\bar{\mathcal{T}},\varphi) = \sum_{i=1}^{|\bar{\mathcal{T}}|} p(t(\mathcal{T}_i)|s(\mathcal{T}_i))\,, \tag{9.11}$$

$$\mathcal{H}_{p(x|y)}(\bar{\mathcal{T}},\varphi) = \sum_{i=1}^{|\bar{\mathcal{T}}|} p(s(\mathcal{T}_i)|t(\mathcal{T}_i))\,, \tag{9.12}$$

donde la probabilidad de traducción directa $p(x|y)$ e inversa $p(y|x)$ se definen como sigue:

$$p(y|x) = \frac{\mathcal{C}(\mathcal{L}(x,y))}{\sum_{\mathcal{L}(x,y')\in\Delta} \mathcal{C}(\mathcal{L}(x,y'))}\,, \tag{9.13}$$

$$p(x|y) = \frac{\mathcal{C}(\mathcal{L}(x,y))}{\sum_{\mathcal{L}(x',y)\in\Delta} \mathcal{C}(\mathcal{L}(x',y))}\,, \tag{9.14}$$

siendo $\mathcal{C}(z \in \Delta)$ la cuenta del número de veces que apareció la tupla z en el corpus bilingüe de entrenamiento, y $\mathcal{L} : \Sigma^+ \times \Omega^* \to \Delta$ el morfismo introducido en la sección 9.4.

Penalización del número de tuplas y palabras

Sirve para penalizar la longitud de la respuesta del sistema (WIP de "Word Insertion Penalty" y TIP de "Tuple Insertion Penalty"), definidos sobre el número de palabras y número de tuplas producidas durante la búsqueda:

$$\mathcal{H}_{wip}(\bar{\mathcal{T}},\varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} \exp\left(|t(\mathcal{T}_i)|\right) = \exp\left(|\bar{y}|\right)\,, \tag{9.15}$$

$$\mathcal{H}_{tip}(\bar{\mathcal{T}},\varphi) = \exp\left(|\bar{\mathcal{T}}|\right)\,. \tag{9.16}$$

Penalización del número de traducciones desconocidas

Sirve para controlar la penalización de las traducciones desconocidas:

$$\mathcal{H}_{uip}(\bar{\mathcal{T}},\varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} \exp\left(\delta(t(\mathcal{T}_i), \mathrm{UNK})\right)\,. \tag{9.17}$$

donde $\delta(\cdot,\cdot)$ es la delta de Kronecker, y UNK hace referencia al token especial palabra desconocida.

9.5.2. Modelos contextuales

Los modelos contextuales son aquellos que para calcular su valor sobre la tupla $z \in \Delta$ necesitan parte del contexto de dicha tupla. Trabajamos sólo con modelos regulares, con lo que únicamente precisan como contexto las tuplas anteriores a z en la hipótesis.

Modelo de lenguaje bilingüe (modelo de traducción)

Establece la probabilidad de cada una de las tuplas del sistema condicionada a la secuencia de las $N-1$ tuplas anteriores y se utiliza para aproximar la probabilidad conjunta $p(\bar{x},\bar{y})$ de la ecuación (9.1). De este modelo típicamente sólo hay uno, el modelo estadístico. En este trabajo añadiremos un NN LM, ambos combinados log-linealmente siguiendo la ecuación (1.3):

$$\mathcal{H}_{\mathcal{ST}'}(\bar{\mathcal{T}},\varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} p(\mathcal{T}_i|\mathcal{T}_{i-1}\dots\mathcal{T}_{i-N+1})\,, \tag{9.18}$$

siendo N la longitud del N-grama. Este modelo computa la probabilidad de la frase siguiendo el autómata estocástico $\mathcal{ST}'$.

Modelo de lenguaje de la lengua destino

Sirve para puntuar la calidad de la frase resultante en la lengua destino. Calcula la probabilidad de la frase dada como respuesta por el sistema, $p(\bar{y})$. De nuevo, de este modelo podemos tener más de uno. En nuestra aproximación tendremos un modelo de N-gramas estadístico estándar y un modelo conexionista. Ambos son combinados siguiendo la ecuación (1.3):

$$\mathcal{H}_{LM}(\bar{\mathcal{T}},\varphi) = \mathcal{H}_{LM}(t(\mathcal{T}_1)t(\mathcal{T}_2)\dots t(\mathcal{T}_{|T|})) = \tag{9.19}$$

$$= \prod_{i=1}^{|\bar{y}|} p(y_i|y_{i-1}\dots y_{i-N+1})\,, \tag{9.20}$$

siendo N la longitud del N-grama.

9.5.3. Modelos de reordenamiento

Modelo de reordenamiento simple

Este modelo permite puntuar el reordenamiento aplicado a las palabras de la frase de entrada, penalizando el cambio de orden de las palabras de la frase de entrada:

$$\mathcal{H}_r(\bar{\mathcal{T}},\varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} \exp\left(abs(last(i-1)+1-first(i))\right)\,, \tag{9.21}$$

donde $last(i-1) = \max_j \left\{ j \;\middle|\; \varphi(j) = i-1 \right\}$ es la posición en la frase de entrada de la última palabra de la tupla $\mathcal{T}_{i-1}$, y $first(i) = \min_j \left\{ j \;\middle|\; \varphi(j) = i \right\}$ es la posición en la frase de entrada de la primera palabra de la tupla $\mathcal{T}_i$, siendo $last(0) = first(0) = 0$.

Modelo de reordenamiento lexicalizado

El modelo de reordenamiento lexicalizado que utilizamos está inspirado en el que utiliza la herramienta Moses [Koehn et al., 2007]. Este modelo nace motivado por el hecho de que la reordenación de segmentos de palabras es dependiente del segmento que estamos reordenando. Es decir, hay segmentos que son reordenados con una mayor frecuencia. Por ejemplo, el cambio de orden entre adjetivos y nombres entre inglés y español.

Para tener en cuenta este hecho, se estima un modelo de reordenamiento lexicalizado que condiciona la probabilidad del reordenamiento al segmento que estamos evaluando. No obstante, el cálculo de esta probabilidad tiene problemas de dispersión, pues determinados segmentos aparecen muy poco en los datos de entrenamiento.

Para solucionar el problema de la dispersión, se pueden utilizar muchas técnicas diferentes. Más adelante comentaremos algunas propuestas de trabajo futuro que podrían ser muy interesantes, combinando la idea de los modelos conexionistas de lenguaje con el reordenamiento lexicalizado.

El modelo de reordenamiento lexicalizado considera tres tipos de orientación:

- Monótona ("Monotonous"): cuando las posiciones en la frase de entrada de las palabras de dos segmentos se siguen unas de otras de forma monótona.
- Invertida ("Swap"): cuando las posiciones en la frase de entrada de las palabras de dos segmentos, si cambiamos el orden de los segmentos (hacemos que el segundo pase a ser el primero y viceversa), se siguen unas de otras de forma monótona.
- Discontinua ("Discontinuous"): cuando no se da ni el primer ni el segundo caso.

En este trabajo no utilizamos directamente la estimación del modelo de reordenamiento de Moses, si bien usamos una adaptación equivalente para el caso concreto de las tuplas bilingües. En la sección 9.5.5 se comenta una deficiencia del método de extracción de tuplas que hemos tenido en cuenta al especificar este modelo.

Formalmente, dada la función φ, la tupla actual $\mathcal{T}_i$ y la tupla anterior $\mathcal{T}_{i-1}$, la orientación entre ambas tuplas es:

$$o_d = \begin{cases} D, & \text{sii } (\textbf{tuplas solapadas}) \vee (abs(first(i) - last(i-1)) \neq 1)\,, \\ M, & \text{sii } (\textbf{tuplas no solapadas}) \wedge first(i) - last(i-1) = 1\,, \\ S, & \text{sii } (\textbf{tuplas no solapadas}) \wedge first(i) - last(i-1) = -1\,, \end{cases} \tag{9.22}$$

donde M, S y D indica la orientación, que puede ser monótona, invertida o discontinua respectivamente. Diremos que **dos tuplas están solapadas** cuando los intervalos formados por sus respectivos $[first(i-1), last(i-1)], [first(i), last(i)]$ estén solapados. La razón por la que pueden estar solapados se explica en la sección 9.5.5.

La probabilidad de la orientación siendo $t = \mathcal{T}_i$ se define como:

$$p_d(o_d|t) = (1-\beta)\frac{\mathcal{C}(o_d, t)}{\sum_{o'_d} \mathcal{C}(o'_d, t)} + \beta \frac{\mathcal{C}(o_d)}{\sum_{o'_d} \mathcal{C}(o'_d)}\,, \tag{9.23}$$

donde $\beta \in [0, 1]$ es un peso que sirve para combinar el modelo suavizador que evita los problemas de dispersión. En este caso el modelo utilizado para suavizar $\dfrac{\mathcal{C}(o_d)}{\sum_{o'_d} \mathcal{C}(o'_d)}$ es la distribución de probabilidad a priori de o_d.

La probabilidad de la orientación inversa (anterior tupla respecto a la actual) se puede calcular de modo similar. Basta con cambiar en la ecuación (9.22) $first(i)$ por $first(i-1)$ y $last(i-1)$ por $last(i)$, siendo $t = \mathcal{T}_{i-1}$.

Las tres posibles orientaciones y las dos posibles direcciones producen seis modelos lexicalizados de reordenamiento diferentes:

$$\mathcal{H}_{R1}(\bar{\mathcal{T}}, \varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} p_d(\mathcal{T}_i | o_d = M)\,, \tag{9.24}$$

$$\mathcal{H}_{R2}(\bar{\mathcal{T}}, \varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} p_d(\mathcal{T}_i | o_d = S)\,, \tag{9.25}$$

$$\mathcal{H}_{R3}(\bar{\mathcal{T}}, \varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} p_d(\mathcal{T}_i | o_d = D)\,, \tag{9.26}$$

$$\mathcal{H}_{R4}(\bar{\mathcal{T}}, \varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} p_i(\mathcal{T}_{i-1} | o_i = M)\,, \tag{9.27}$$

$$\mathcal{H}_{R5}(\bar{\mathcal{T}}, \varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} p_i(\mathcal{T}_{i-1} | o_i = S)\,, \tag{9.28}$$

$$\mathcal{H}_{R6}(\bar{\mathcal{T}}, \varphi) = \prod_{i=1}^{|\bar{\mathcal{T}}|} p_i(\mathcal{T}_{i-1} | o_i = D)\,. \tag{9.29}$$

La probabilidad de la orientación es 1 en el caso en que no coincida con el parámetro o del modelo.

9.5.4. Modelo conexionista de reordenamiento

Es posible mejorar el modelo anterior si utilizamos un clasificador que permita obtener la probabilidad de la orientación. Dicho clasificador recibiría a su entrada la tupla actual y la anterior. Utilizando la idea de la proyección en el espacio continuo, las tuplas anterior y actual podrían codificarse en una red neuronal que recibiría como entrada la proyección de las tuplas. Esto significaría que necesitaríamos entre 500 y 1000 neuronas en la entrada de la red neuronal. La proyección en el espacio continuo podría estimarse entrenando un modelo conexionista de lenguaje, quedándonos únicamente con la capa de proyección para llevar adelante este cometido. La combinación lineal de más de un modelo ayudaría a la generalización. El modelo computaría a su salida la probabilidad de cada una de las orientaciones dadas.

9.5.5. Deficiencias en el modelado

Las deficiencias, o problemas del modelado, aparecen debidas a la sinergia existente entre el formalismo matemático que estamos siguiendo y el algoritmo de decodificación utilizado. Estos problemas se resuelven *siempre* antes de entrenar los modelos de la combinación loglineal, es decir, los modelos de nuestro sistema se entrenan sobre el corpus debidamente preprocesado para dar solución a estos problemas.

Figura 9.5: *Ejemplo de alineamiento entre palabras que conlleva una extracción de tuplas bilingües* deficiente*. //* Instance of a deficient tuples extraction.

Discontinuidades en la parte origen de las tuplas

La técnica del unfolding permite reducir el vocabulario de tuplas, ya que genera tuplas más reutilizables en el sistema de traducción [Crego *et al.*, 2005c], pero a su vez introduce una deficiencia en el modelado, la aparición de discontinuidades en la parte origen de las tuplas. Como ya se ha comentado anteriormente esto puede ser problemático ya que implica que las palabras de la parte origen de la tupla no están ordenadas de forma monótona (aunque aseguramos que sí lo estén de forma creciente para aliviar el problema del reordenamiento). Debido a esta problemática aparecen ambigüedades en los modelos de reordenamiento de la sección 9.5.3 ya que estos modelos calculan su coste a partir de la posición en la frase de entrada de la primera y la última palabra de la parte origen de las tuplas. La figura 9.5 ilustra un ejemplo de este problema. El alineamiento de las palabras, si se traduce en la dirección Inglés-Español, fuerza la aparición de dos tuplas iniciales que son:

also#También I_shall_refer#someteré ...

de manera que el segmento *also* aparece en la posición 3 de la frase de entrada y el segmento *I_shall_refer* tiene como posición inicial y final la 1 y la 4 respectivamente. Para el modelo lexicalizado de reordenamiento estas dos tuplas estarán en orientación *discontinua*. También sería posible solucionar este problema añadiendo un nuevo tipo especial de orientación entre tuplas.

Palabras empotradas

Por otro lado, la segmentación única del corpus bilingüe no asegura que todas las palabras del vocabulario origen Σ aparezcan en tuplas con longitud uno en la parte origen ($|s(z \in \Delta)| = 1$). Esto implica que para traducir dichas palabras será necesario que aparezcan con el mismo contexto que alguna de las tuplas que las contienen. A estas palabras se les llama palabras *empotradas* [Crego *et al.*, 2006b]. Tendremos tres formas de solucionar este problema, que consisten en ampliar el vocabulario de tuplas Δ con:

a) Tuplas donde la parte origen son las palabras empotradas, y cuya parte destino es la palabra desconocida, y que todos los modelos de la combinación log-lineal tratarán como palabras desconocidas.

b) Tuplas del diccionario léxico directo e inverso calculado a partir del alineamiento (ecuaciones (9.13) y (9.14)), asociando a cada palabra empotrada la palabra destino

que maximiza la probabilidad combinada de ambos diccionarios. El modelo bilingüe de traducción tratará estas tuplas como desconocidas, sin embargo, sí que podemos dar probabilidad a estas tuplas usando los modelos incontextuales.

c) Extraer tuplas del diccionario léxico directo e inverso el conjunto de las K mejores asociaciones entre la palabra *empotrada* y palabras de la lengua destino, en lugar de quedarnos únicamente con la mejor. Esto permitiría que el modelo de lenguaje también participara en la decisión de elegir la mejor asociación. Se puede considerar una generalización del caso (b) que dejamos como trabajo futuro.

Palabras desconocidas y el reordenamiento

Nuestra tercera deficiencia se debe a un efecto combinado de las discontinuidades y de las palabras empotradas cuando se usa la aproximación (a). En determinadas situaciones puede suceder que tenga mayor puntuación la traducción de una palabra como palabra desconocida, y sin embargo puede existir un reordenamiento de las palabras que permita traducirla de forma correcta mediante una tupla de $|s(z \in \Delta)| > 1$. Para tratar de aliviar este problema se introducirá un factor de penalización en el sistema que castigará los caminos donde haya palabras traducidas como desconocidas. Dicha penalización podrá ser fijada a priori, o bien podrá estimarse dentro del marco de máxima entropía como un modelo más de la combinación log-lineal. En caso de usar la aproximación (b), esta penalización no tiene efecto en la búsqueda.

Palabras de la frase de salida sin alineamiento en la entrada

Existe una cuarta deficiencia. Aquellas palabras de la frase de salida que se alinean con `<NULL>`, son consideradas como inserciones en la literatura [Knight & Al-Onaizan, 1998; Bangalore & Riccardi, 2000]. No obstante, esto implica un mayor coste del algoritmo de decodificación. Siguiendo el trabajo de [Mariño *et al.*, 2006], en esta tesis evitaremos la aparición de inserciones en el algoritmo de decodificación. Debido a esto, una vez han sido extraídas todas las tuplas y el corpus ha sido segmentado, se lleva a cabo un procesamiento del corpus que obliga a asociar dichas palabras con la tupla que queda a su derecha o a su izquierda. Se escoge la tupla con mayor probabilidad combinada de las ecuaciones (9.6) y (9.7) tras añadir la palabra en cuestión. En la literatura se han probado diferentes criterios [Mariño *et al.*, 2006], si bien no parece haber grandes diferencias entre ellos.

Reducción del diccionario de tuplas

Finalmente, el tamaño del diccionario de tuplas extraído es un problema si es demasiado grande, derivando en una decodificación demasiado lenta. Para aliviarlo, se restringe el número de tuplas que comparten un mismo segmento de frase de entrada [Mariño *et al.*, 2006]. Se escoge un valor n para dicha restricción quedándonos siempre con las n tuplas más frecuentes de cada conjunto que comparte el mismo segmento de la frase de entrada.

9.6 Optimización de los pesos de la combinación log-lineal

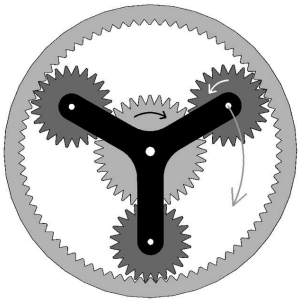
Para llevar a cabo la optimización de los pesos de la combinación log-lineal es posible utilizar el algoritmo MERT como se ha explicado en la sección 1.7. Este algoritmo implementa una búsqueda del conjunto de valores para los pesos que minimiza el BLEU de un conjunto de evaluación. Es muy flexible, ya que admite cualquier número de modelos a combinar, si bien conforme crece el número de modelos, aumentan los problemas de convergencia del algoritmo. En traducción automática, Moses [Koehn et al., 2007] es una herramienta estándar cuyos resultados son estado del arte. Por ello optimizamos los pesos de la combinación utilizando la implementación del algoritmo de Powell [Powell, 1964] distribuido libremente con Moses.

9.7 Resumen

Este capítulo ha introducido las bases formales de la traducción automática estadística, y concretamente la aproximación basada en N-gramas. Ha explicado los modelos que es posible utilizar en la combinación log-lineal y que serán desarrollados por el sistema de traducción resultado de esta tesis. También se han comentado las deficiencias más importantes de esta aproximación y cómo han de solucionarse para poder entrenar los modelos aquí descritos.

10

ALGORITMO DE DECODIFICACIÓN PARA TRADUCCIÓN AUTOMÁTICA



Índice

En este capítulo se explican los algoritmos que se han desarrollado para abordar tareas de traducción automática estadística. Se ha diseñado un sistema completo que realiza una búsqueda de la mejor traducción aplicando poda estática y poda en haz para reducir la complejidad del problema. También incorpora un sistema de generación de hipótesis de reordenamiento de las palabras de la frase de entrada. Este sistema de reordenamiento puede restringir las hipótesis generadas mediante los heurísticos más utilizados en el campo de la traducción automática.

El objetivo de este sistema es doble:

- mostrar la generalidad de los algoritmos de Viterbi y de redes neuronales desarrollados para reconocimiento de secuencias, y como estos pueden adaptarse a la traducción automática;
- permitir la incorporación de NN LMs en el proceso de traducción de forma totalmente acoplada.

Todos los algoritmos son reactivos, es decir, funcionan ejecutando acciones como respuesta a peticiones que reciben. Todos los mensajes que intercambian las distintas partes en que se ha dividido el sistema están relacionados con el envío de grafos siguiendo el protocolo de incidencia de la sección 1.4.1. El algoritmo que aquí se describe está basado en el algoritmo de dos pasos del campo del reconocimiento de habla. El proceso de segmentación de la señal en palabras, necesario para el reconocimiento de voz, se interpreta aquí como un proceso de segmentación de la frase de entrada (probablemente reordenada) en segmentos que son secuencias de palabras que traducimos todas a la vez, basándonos en el paradigma de la traducción basada en segmentos.

10.1 Descripción general del procedimiento de búsqueda

El sistema de traducción implementado es una adaptación del algoritmo de dos pasos utilizado para tareas de reconocimiento de escritura/habla, descrito en el capítulo 6. En este caso todos los módulos intercambian información en ambas direcciones: emiten un grafo y reciben como retroalimentación la mejor probabilidad del vértice activo. También se utilizan heurísticos para calcular el coste futuro de un nodo, de tal forma que dicho heurístico junto con la retroalimentación permite podar de forma más robusta la lista de estados activos de los algoritmos implementados en cada módulo.

La figura 10.1 ilustra los módulos utilizados en el sistema de traducción automática:

- Módulo de reordenamiento: genera un grafo de reordenamiento siguiendo una serie de heurísticos y podando dicho grafo con la información de las etapas posteriores y un heurístico de coste futuro de cada nodo. Este módulo genera un grafo multietapa, y expande una etapa entera antes de aplicar la poda.

- Módulo Word2Tuple: es el encargado de convertir el grafo de reordenamiento en un grafo de tuplas. A grandes rasgos aplica una clausura transitiva al grafo que recibe por su entrada y en las aristas pone listas de tuplas que podrían haberse traducido por el conjunto de palabras de la lengua origen que cubre dicha arista. Este módulo aplica los modelos incontextuales de las tuplas: probabilidades condicionales directa e inversa, modelos léxicos tipo IBM-1 directo e inverso, penalización por número de tuplas, penalización por número de palabras en la lengua destino. En este caso se envía la lista de incidencia completa de un vértice y luego espera a recibir la información de retroalimentación del siguiente módulo, que indicará cual ha sido la mejor probabilidad de la búsqueda encontrada llegando hasta ese vértice.

- Módulo Viterbi: recibe el grafo y va calculando conforme lo recibe el camino de mayor probabilidad. Es el módulo final y aplica las probabilidades de los modelos contextuales (modelo de lenguaje destino y modelo de traducción bilingüe) y de los modelos de reordenamiento. Este módulo genera un trellis completo de la búsqueda, de tal forma que una vez finalizado el proceso es posible obtener tanto el mejor camino, como los N mejores. Para la lista de N-best se utiliza una implementación eficiente del algoritmo de Eppstein [Eppstein, 1998; Jiménez & Marzal, 2003].

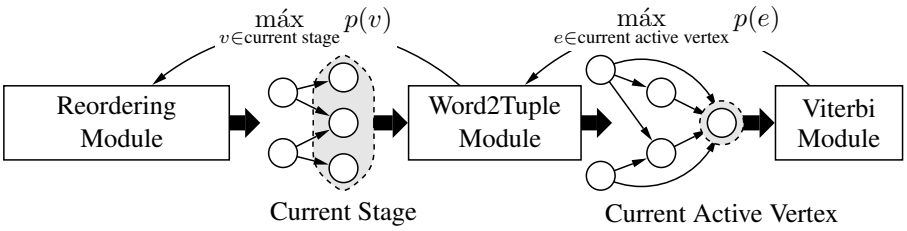


Figura 10.1: *Diagrama de bloques de la arquitectura* `dataflow` *para traducción. //* `Dataflow` diagram for machine translation. Referenced at page 313.

Toda esta arquitectura, hasta la búsqueda de las N-best, ha sido implementada dentro del toolkit `April`, que ya hemos mencionado anteriormente, y cuya existencia ha sido muy importante para el desarrollo de esta y otras tesis. La filosofía modular de dicho toolkit permite tener una gran flexibilidad a la hora de añadir o quitar módulos de la arquitectura y probar diferentes algoritmos. Otra de las ventajas de este tipo de arquitectura es la fácil integración con procesos tipo on-line. Por ejemplo, es posible modificar ligeramente la arquitectura y tener un sistema de traducción de habla/escritura on-line.

10.1.1. Preliminares

Vamos a formalizar algunas estructuras de datos utilizadas en la implementación de los módulos. Para representar tuplas bilingües se ha utilizado un modelo de estados finitos, concretamente un transductor con forma de árbol de prefijos. Formalmente es una tupla $\mathcal{A} = \langle \mathcal{Q}, q_R, \Sigma, \Delta, \delta, P_I, P_T, P_E \rangle$, donde:

- $\mathcal{Q} = \{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$ es el conjunto de nodos del árbol de prefijos;
- $q_R \notin \mathcal{Q}$ es el nodo raíz del árbol;
- Σ es el vocabulario de palabras de la lengua origen;
- Δ es el vocabulario extendido mediante GIATI, y por lo tanto, es el vocabulario de tuplas bilingües del sistema;
- $\delta : (\mathcal{Q} \cup q_R) \times \Sigma \rightarrow \mathcal{Q}$ es la función de transición del transductor;
- $P_I \in \mathbb{R}^{\geq 0}$ es la probabilidad de ser inicial el estado q_R, que se calcula al adelantar probabilidades en el modelo;
- $P_T : (\mathcal{Q} \cup q_R) \times \Sigma \rightarrow \mathbb{R}^{\geq 0}$ es la probabilidad de transitar, que también se calcula al adelantar probabilidades;
- $P_E : \mathcal{Q} \times \Delta \rightarrow \mathbb{R}^{\geq 0}$ es la probabilidad de que un determinado nodo emita una tupla. Esto es, que la secuencia de palabras que llega hasta un nodo determinado se traduce en la tupla indicada, con una determinada probabilidad. Esta probabilidad se calcula a partir de la combinación log-lineal de los modelos incontextuales (véase sección 9.5.1).

Es importante destacar que en este caso no estamos ante un modelo estocástico, ya que las funciones P_T y P_E son el resultado de una combinación log-lineal de varios modelos, y por tanto no son distribuciones de probabilidad normalizadas.

Bajo esta formalización, podemos construir el árbol de prefijos mediante este proceso:

- Cada nodo del árbol será identificado con la secuencia de palabras en la lengua origen que llega hasta él. Por tanto un nodo q_i se representará como una secuencia de longitud $j \geq 0$ de palabras de Σ. $q_i = q_{R,x_1,x_2,\ldots,x_j}$. La primera siempre será R que identifica al nodo raíz (q_R).
- De esta forma la función de transición consiste en ir al nodo cuyo nombre se identifica con la concatenación de la siguiente palabra:

$$\delta(q_{R,x_1,\ldots,x_j}, w \in \Sigma) = q_{R,x_1,\ldots,x_j,w} \,.$$

- Inicialmente $P_I = 1$ y $P_T(q_i, w) = 1 \ \forall \ \delta(q_i, w)$, y

$$P_E(q_{R,x_1,\ldots,x_j}, z \in \Delta) = \begin{cases} 0, & \text{sii } x_1, \ldots, x_j \neq s(z); \\ \prod_{i=1}^{M'} \mathcal{H}_i(z)^{\lambda_i}, & \text{sii } x_1, \ldots, x_j = s(z); \end{cases}$$

  donde M' es el número de modelos incontextuales de la combinación log-lineal, λ_i su coeficiente de mezcla y $\mathcal{H}_i(z)$ la puntuación del modelo i.

Para estimar las probabilidades de ser inicial P_I y de transitar P_T se realiza un procedimiento de adelantamiento de probabilidades en el cual se modifica la probabilidad de emitir P_E. Adelantar las probabilidades permite tener una información más completa en caso de querer aplicar poda en haz o poda estática. El proceso seguido para adelantar las probabilidades es un proceso recursivo y se puede formalizar de esta forma:

$$P'_E = P_E \tag{10.1}$$

$$L(q_i \in \mathcal{Q}) = \text{máx}\{\max_{z \in \Delta} P'_E(q_i, z), \max_{q' \in \{\delta(q_i, w) \forall w \in \Sigma\}} L(q')\} \tag{10.2}$$

$$P_I = L(q_R) \tag{10.3}$$

$$P_T(q_i \in \mathcal{Q}, w \in \Sigma) = \frac{L(\delta(q_i, w))}{L(q_i)} \tag{10.4}$$

$$P_E(q_i \in \mathcal{Q}, z \in \Delta) = \frac{P'_E(q_i, z)}{L(q_i)} \tag{10.5}$$

La figura 10.2 ilustra un ejemplo de transductor árbol de prefijos con todas sus probabilidades estimadas. Dado un camino, por ejemplo abd, la puntuación de dicho camino es el resultado de multiplicar la puntuación de las aristas, P_T, por la puntuación del nodo raíz, P_I, y la de emitir de cada tupla, P_E:

- $abd|||EBC$ con puntuación $0.9 \cdot 0.2 \cdot 0.3 \cdot 0.5 \cdot 0.4 = 0.1080$,
- $abd|||D$ con puntuación $0.9 \cdot 0.2 \cdot 0.3 \cdot 0.5 \cdot 0.2 = 0.0054$.

Los nodos se identifican con un índice numérico (0 para el nodo q_R) que ha sido representado dentro del diagrama de cada nodo. Junto al índice se ha representado la lista de tuplas con probabilidad de emisión P_E diferente de 0. Cada tupla se ha representado con tres partes separadas por $|||$:

- la primera es la secuencia de palabras en la lengua origen (la secuencia que nos ha llevado hasta ese nodo);
- la segunda es la secuencia de palabras en la lengua destino (la traducción de la primera);
- la tercera es la puntuación o probabilidad de emitir de dicha tupla, $P_E(q_i, z)$.

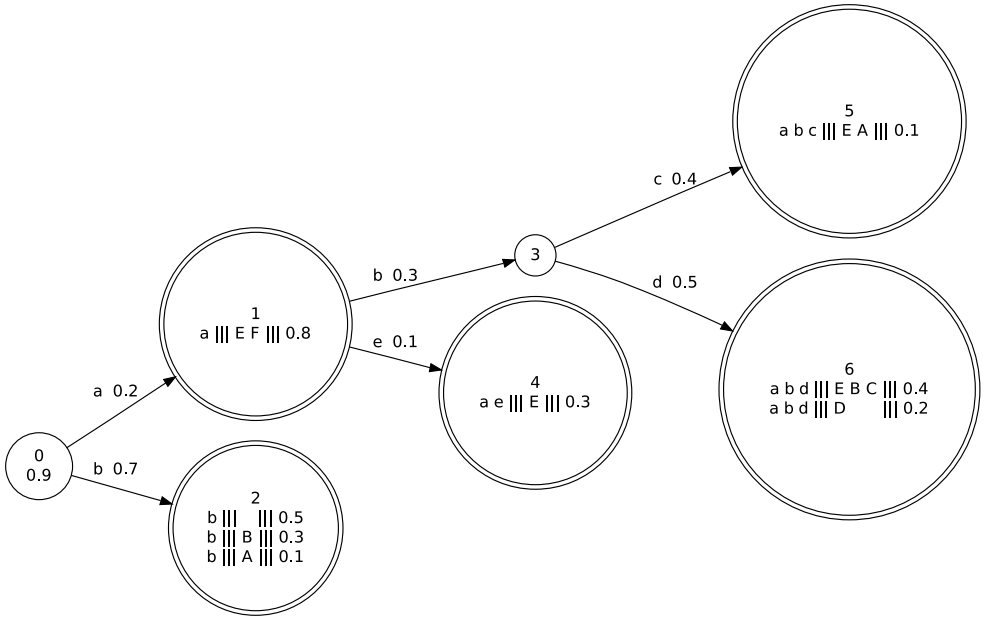


Figura 10.2: *Ejemplo de árbol de prefijos transductor. //* Instance of a transducer trie.

10.2 Generación del grafo de reordenamiento

En esta etapa el sistema genera un grafo de reordenamiento que servirá de punto de partida para el resto de etapas del sistema. Para generar dicho grafo se sigue el esquema del algoritmo de programación dinámica para resolver el problema del viajante de comercio. Este esquema permite reducir la complejidad de la generación del grafo de $O(|\bar{x}|!)$ a $O(2^{|\bar{x}|})$. La complejidad del resto del sistema de traducción depende de la complejidad de esta etapa, y por lo tanto, aun a pesar de la drástica reducción ofrecida por el algoritmo de programación dinámica, sigue siendo demasiado *caro* generar todos los posibles reordenamientos.

Muchos autores han presentado trabajos en los que se estudia la aplicación de reglas a la generación de este tipo de grafos [Tillmann & Ney, 2003; Crego *et al.*, 2005c, 2006b], generando sólo los caminos más prometedores. Aunque el proceso de generación del grafo de reordenamiento se realiza en una etapa desacoplada, los resultados obtenidos son realmente buenos y permite aumentar en gran medida la eficiencia de todo el sistema. Este tesis adopta una solución general que permite construir dicho grafo a partir de información sobre la calidad de la traducción obtenida por las siguientes etapas del sistema. Para este cometido, es necesario conocer para cada nodo del grafo de reordenamiento cual es la mejor puntuación obtenida por el resto del sistema de traducción, y combinar dicha puntuación con una cota optimista de lo bueno que puede llegar a ser el camino que sigue a partir de dicho nodo. Se ha implementado el cálculo del coste futuro de un nodo siguiendo la aproximación de Moses. En esta etapa se aplica poda en haz de tipo dinámico y estático, utilizando como información extra el coste futuro. Aun así, no podremos asegurar que el camino de mejor probabilidad sobreviva a dicha poda. El grafo generado por este módulo es un grafo multietapa, donde las aristas se etiquetan con posiciones de las palabras de la frase de entrada $\bar{x}$. Cada etapa

corresponde a haber procesado una palabra de entrada más, con lo que el grafo tiene $|\bar{x}| + 1$ etapas. El tamaño de cada etapa depende de los parámetros de la poda y el heurístico elegido para expandir posiciones no cubiertas de la frase de entrada. No obstante, la etapa se genera completa, siguiendo el heurístico elegido, y la poda se aplica tras expandir, cuando se conoce la probabilidad que asocia el sistema de traducción a cada uno de los nodos que se han generado en la etapa actual.

El algoritmo tiene tres parámetros que controlan la aplicación de las podas:

- haz estático: que servirá para poner un límite en el número de vértices que serán expandidos en cada etapa;
- haz dinámico inicial: indica el valor del haz dinámico en la primera etapa de programación dinámica;
- haz dinámico final: indica el valor del haz dinámico en la última etapa de programación dinámica.

En las etapas intermedias (entre la inicial y la final) se aplica un haz dinámico que es la interpolación lineal correspondiente en base a los valores inicial y final.

El algoritmo no genera probabilidades, pues no tiene ningún tipo de modelo con el que generarlas, sin embargo, sí que utiliza la probabilidad del sistema de traducción y el coste futuro de cada nodo para guiar la búsqueda. Por consiguiente, al finalizar la expansión de cada etapa del grafo, el algoritmo espera a que el resto del sistema produzca sus hipótesis, para recibir la información de cual ha sido la mejor hipótesis encontrada en cada uno de los vértices generados en la etapa actual. La espera se realiza en la línea 31 (del algoritmo 10.1), punto donde se sincroniza con el resto del sistema de traducción, y recibe una lista con las mejores probabilidades encontradas para los nuevos estados generados. El algoritmo 10.1 formaliza este proceso. [1]

Este algoritmo se puede extender de forma muy sencilla para dar cuenta de lo que se conoce en traducción automática como *paredes* (o *walls*). Las paredes son palabras en la frase origen que obligan a que todo lo que les precede haya sido traducido antes de comenzar a traducir las palabras que suceden a la palabra que hace de pared. Esta extensión ha sido implementada en el sistema y se utiliza por defecto aplicando *paredes* en los signos de puntuación:

```
. , : ; - -- ? !
```

10.2.1. Heurísticos para generar el grafo de reordenamiento

Como ya se ha comentado, el coste de generar todas las posibles formas de ordenar las palabras de la frase de entrada, incluso aplicando técnicas de programación dinámica, es de $O(2^{|\bar{x}|})$ siendo $|\bar{x}|$ el número de palabras de la frase. Este coste es inadmisible para cualquier

[1] Se deja como trabajo futuro el diseño de un algoritmo basado en reglas que construya un grafo de reordenamiento más restringido, lo cual permitirá acelerar la latencia total del sistema. Para ello, tan sólo habría que modificar ligeramente el algoritmo 10.1 o bien implementar uno nuevo que siguiera sincronizando con el sistema completo de traducción la generación y expansión del grafo. Esto permitiría generar el grafo en base a dichas reglas, y la probabilidad (o puntuación) de las reglas podría usarse como una fuente de información adicional para la combinación log-lineal.

Algoritmo 10.1 Genera un grafo de reordenamiento. // *It generates a word-reordering graph. Referenced at page 314.*

Require: A source sentence size $|\bar{x}|$, reordering constraints, max histogram pruning size, bw_{begin} and $b_w end$ initial and final beam search values

Ensure: Produce a reordering graph constrained to input parameters

1: **procedure** reorderingGraphGeneration($|\bar{x}|, ReordConstraints, max, bw_{begin}, bw_{end}$)
2: **begin**
3: Let initial vertex $v = 0$
4: Let active states hash $C = \{(\{1, 2, \ldots, |\bar{x}|\}) \rightarrow (0.0, v)\}$ // The active states list is initiated with only one hypothesis, which key is the set of all possible source word positions (uncovered source word positions), and its value the pair of probability and vertex index
5: Generate message `vertex` (v)
6: Generate message `is_initial` (1)
7: **for** $i = 1, \ldots, |\bar{x}|$ **do**
8: Let L a list of dataflow messages to be sended at the end
9: Let $C' = \emptyset$
10: Apply future cost for uncovered word positions k to probability p of every hypothesis $(k) \rightarrow (p, u) \in C$
11: Let $best =$ the best hypothesis probability stored at C
12: Compute dynamic beam-width as $beamwidth = bw_{end} \frac{bw_{begin}}{bw_{end}}^{\frac{|\bar{x}|-i-1}{|\bar{x}|-1}}$
13: Let $processed = 0$, the number of processed vertices
14: **for all** hypothesis $(k) \rightarrow (p, u) \in C$ ordered by p, in descending order, until $processed > max$ or $(\frac{best}{p} > beamwidth)$ **do**
15: $processed++$
16: **for all** word position $j \in k$ which acomplish the $ReordConstraints$ **do**
17: Let $k' = k$ removing word position j from it
18: Let $v' =$ vertex asocciated with k' key in C'
19: **if** v' is **nil then**
20: Let $v' = v$
21: $v = v + 1$
22: Store message `vertex` (v') at L
23: Insert at C' the key-value pair $(k') \rightarrow$ (Unknown probability, v')
24: **end if**
25: Store message `edge` (v', data=$\{\langle j, 1.0\rangle\}$) at L
26: **end for**
27: **end for**
28: Generate dataflow messages from list L in a topologic order
29: Generate message `no_more_out_edges` for each vertex hypothesis on C
30: Generate message `no_more_in_edges` for each vertex hypothesis on C'
31: Wait for one `score_feed_back` message for each hypothesis on C' and update probability with each returned value
32: Generate message `change_stage`
33: Let $C = C'$
34: **end for**
35: **end procedure**

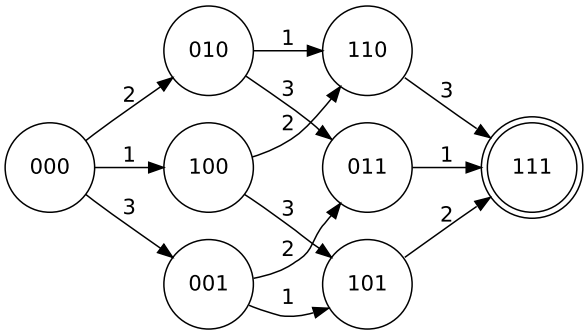


Figura 10.3: *Ejemplo de grafo producido por la etapa de reordenamiento para una frase con tres palabras. Las aristas se etiquetan con la posición de la palabra en la frase de la lengua origen. Los nodos contienen el vector de bits que indica las palabras que ya han sido cubiertas para llegar al nodo.* // Graph generated at the word-reordering stage.

valor relativamente pequeño de $|\bar{x}|$, y es más, de ese enorme conjunto de posibilidades sólo un porcentaje muy pequeño son ordenaciones que pueden dar lugar a traducciones coherentes.

En la literatura la propuesta más generalizada para solventar los problemas computacionales es restringir la búsqueda mediante algún tipo de heurístico que sólo permita algunas ordenaciones que a priori son plausibles [Berger *et al.*, 1996b; Tillmann & Ney, 2003; Zens *et al.*, 2004; Matusov *et al.*, 2005; Kanthak *et al.*, 2005].[2]

Los heurísticos más extendidos [Berger *et al.*, 1996b; Matusov *et al.*, 2005], y que han sido implementados en el módulo de reordenamiento, son estos tres:

- Restricción IBM-l: la idea que hay detrás de esta restricción consiste en desviarse de la traducción monótona posponiendo la traducción de algunas pocas palabras. Se implementa cubriendo siempre las primeras l palabras que haya sin cubrir.

- Restricción IBM-inverse-l: para algunas traducciones entre lenguas es mejor traducir primero una pequeña parte del final de la frase para después continuar casi de forma monótona. Sea j la primera posición que todavía no ha sido cubierta. Este método se implementa escogiendo cualquier posición no cubierta j' hasta que que las $l-1$ posiciones $j' > j$ hayan sido traducidas. Si se produce este caso, debemos traducir obligatoriamente la palabra j.

- Restricción local con ventana l: se inspira en la traducción entre lenguas donde el reordenamiento sólo afecta a mover grupos pequeños de palabras unas pocas posiciones a la derecha o a la izquierda. En este caso, sólo se permite escoger para traducir posiciones no cubiertas que estén en una ventana de tamaño l alrededor de la primera posición no cubierta.

La figura 10.4 ilustra ejemplos de los heurísticos que acabamos de comentar. El uso de uno u otro heurístico dependerá del par de lenguas que se estén traduciendo, y el tamaño de

[2]Otros autores aprenden reglas de reordenación a partir del corpus de entrenamiento [Tillmann & Ney, 2003; Crego *et al.*, 2006b]. Esta alternativa no vamos a tenerla en cuenta en el desarrollo de esta tesis, no obstante, al trabajar con módulos independientes donde cada uno produce y procesa un grafo, bastaría con desarrollar un módulo capaz de generar *al vuelo* el grafo de entrada utilizando dichas reglas, sin modificar el resto del sistema.

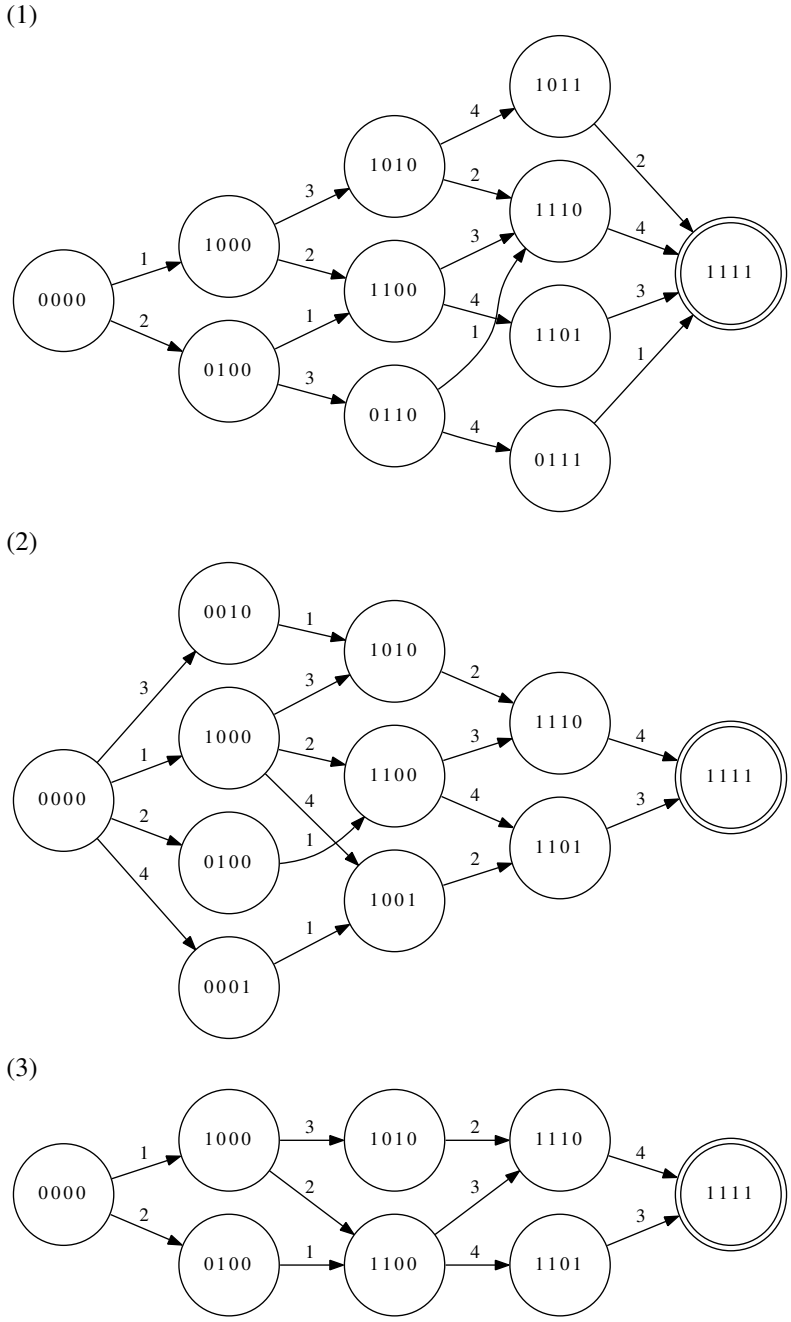


Figura 10.4: *Ejemplos de grafos de reordenamiento siguiendo los heurísticos (1) IBM, (2) IBM-inverse, (3) local, y un tamaño de ventana $l = 2$.* // Instances of word-reordering graphs using heuristics (1) IBM, (2) IBM-inverse, (3) local. Referenced at page 314.

la ventana l puede ser escogido por prueba y error, aunque dada su dependencia del par de lenguas a traducir, es posible establecer valores que funcionen bien para cada par de lenguas.

10.2.2. Cálculo del coste futuro

Para el cálculo del coste futuro de un nodo del grafo de reordenamiento vamos a utilizar la aproximación desarrollada por P. Koehn en Moses [Koehn et al., 2007]. Simplificaremos el problema de tal forma que asumiremos que la traducción de las palabras que faltan por procesar se hará de forma monótona. Se implementará un algoritmo que por programación dinámica calcule el resultado de dicho coste futuro.

En el cálculo del coste futuro se utilizará solamente la información del modelo de lenguaje, del modelo de traducción $\mathcal{ST}'$ y del conjunto de modelos combinados en el transductor árbol de prefijos $\mathcal{A}$. La puntuación del modelo de lenguaje sólo se calcula para las palabras que se producen en una tupla, el contexto entre tuplas se omite, y en el caso de $\mathcal{ST}'$ sólo se utiliza la información relativa a los unigramas (siempre estamos en el estado inicial del modelo). Por lo tanto la aplicación de los modelos de lenguaje se realizará sin tener en cuenta el contexto entre las tuplas. Cabe destacar que dada una frase de entrada de longitud $|\bar{x}|$, existen $\frac{|\bar{x}|(|\bar{x}|+1)}{2}$ diferentes secuencias monótonas de palabras. Se precalculará en una tabla el coste futuro de toda posible secuencia monótona.

Durante el proceso de generación del grafo de reordenamiento, cuando se necesita calcular el coste futuro de un nodo, se conoce el conjunto $k = \{k_1, k_2, \ldots, k_n\}$, con $n \leq |\bar{x}|$, de palabras que todavía no han sido traducidas. Si ordenamos dicho conjunto de menor a mayor, podemos extraer todas las secuencias monótonas que es posible formar con dicho conjunto, buscar el coste futuro de cada una de esas secuencias, y devolver como coste futuro de todo el conjunto k el producto (o suma en escala logarítmica) del coste de cada secuencia monótona. Este coste futuro se aplica junto al mejor coste estimado por el resto del sistema para dicho nodo, y se utiliza el resultado final como criterio para decidir que nodos son más prometedores.

Para el cálculo de los costes futuros sólo utilizaremos modelos de lenguaje estadísticos ya que necesitamos calcular el coste de secuencias de palabras *sin* tener en cuenta su contexto y el modelo conexionista *siempre* necesita dicho contexto. Es posible inicializar los modelos estadísticos en el estado q_0, de tal forma que la primera de las probabilidades que se aplique será del unigrama, la segunda la del bigrama, ... El modelo de lenguaje destino se aplica sobre la parte destino de cada tupla, sin tener en cuenta el cambio de contexto al cambiar de tupla, y del modelo bilingüe de traducción sólo se utilizan los unigramas. El algoritmo 10.2 formaliza esta descripción.

Para calcular el coste del algoritmo tendremos en cuenta estos detalles:

- Asumimos un valor máximo de traducciones por cada conjunto de palabras origen de una tupla. Usaremos P para representar este número, y
- asumimos que el coste de calcular la probabilidad del modelo de lenguaje es despreciable.

El coste queda así:

$$O(|\bar{x}|^2 + |\bar{x}|^2 \cdot (|\Delta| + |\bar{x}|)) = O(|\bar{x}|^2 + |\bar{x}|^2 \cdot (P + |\bar{x}|)) = O(|\bar{x}|^2 P + |\bar{x}|^3),$$

Algoritmo 10.2 Precomputa el coste futuro de las $\frac{|\bar{x}|(|\bar{x}|+1)}{2}$ secuencias de palabras diferentes, dada la frase de entrada. // *It computes the future cost of the* $\frac{|\bar{x}|(|\bar{x}|+1)}{2}$ *different word sequences, given an input source sentence. Referenced at page 314.*

Require: Source sentence words sequence $\bar{x}$, tuples prefix tree $\mathcal{A}$, target language model tlm and stochastic translation model $\mathcal{ST}'$

Ensure: A matrix with all possible future costs

1: **function** futureCostComputation($\bar{x} = \{x_1, \ldots, x_{|\bar{x}|}\}, \mathcal{A}, tlm, \mathcal{ST}'$)
2: **begin**
3: Let $F = |\bar{x}| \times |\bar{x}|$ all-zeroes triangular matrix
4: **for all** begining source word position $b = 1, \ldots, |\bar{x}|$ **do**
5: Let $st = q_R$ the initial state of the prefix tree $\mathcal{A}$
6: Let $p = P_I$ the initial probability of the hypothesis
7: **for all** ending source word position $e = b, \ldots, |\bar{x}|$ **do**
8: **if** $\exists$ transition $\delta(st, x_e)$ at $\mathcal{A}$ **then**
9: Let $st' = \delta(st, x_e)$ the destination state at $\mathcal{A}$
10: Let $p = p \cdot P_T(st, x_e)$ the result of the transition probability
11: **for all** possible tuple $z \in \Delta$ which emission probability $P_E(st', t) > 0$ **do**
12: Let $p_{\mathcal{ST}'}$ = unigram probability of translation model $\mathcal{ST}'$ for tuple z
13: Let p_{tlm} = language model tlm probability for target words $t(z)$
14: Let $p' = p \cdot p_{\mathcal{ST}'}^{\lambda_{\mathcal{ST}'}} \cdot p_{tlm}^{\lambda_{tlm}} \cdot P_E(st', z)$
15: $F(b, e) = \max\{F(b, e), p'\}$
16: **end for**
17: **for all** previous source word positions $k = b - 1$ descending to 1 **do**
18: $F(k, e) = \max\{F(k, e), F(k, b-1) \cdot F(b, e)\}$
19: **end for**
20: Let $st = st'$
21: **else**
22: break
23: **end if**
24: **end for**
25: **end for**
26: **return** F
27: **end function**

que dependiendo del número $|\bar{x}|$ de palabras en la lengua origen será $O(|\bar{x}|^3)$, o bien $O(|\bar{x}|^2 P)$. Es un algoritmo de coste cúbico, dependiendo del valor de $|\bar{x}|$ en el peor de los casos. Dado que normalmente $|\bar{x}| < 100$, el coste se traduce en realizar 1 000 000 de operaciones muy rápidas, con lo que en la práctica es totalmente despreciable, menos de 0.05 segundos, e incluso 0.01 si tenemos valores de $P < 40$ y de $|\bar{x}| < 30$.

10.3 Generación del grafo de tuplas: Word2Tuple

Una vez se ha generado el grafo de reordenamiento, le toca el turno a un segundo módulo que se encarga de substituir y añadir aristas con tuplas al grafo que recibe por la entrada. Este módulo recibe el nombre de *Word2Tuple* y hace uso del transductor en forma de árbol de prefijos en el que se representa el vocabulario de tuplas del sistema.

En la figura 10.5 se ha representado el grafo de reordenamiento de entrada y en color gris las aristas resultantes del proceso Word2Tuple. La figura 10.6 representa un ejemplo del conjunto completo de aristas con tuplas que se genera, dado un camino en el grafo de la figura 10.5.

El algoritmo que genera el grafo de tuplas a partir del grafo de reordenamiento está basado en la forma en que se comunican el proceso anterior y este [España-Boquera *et al.*, 2007]. Se sigue un protocolo de comunicación mediante el cual el grafo es transmitido mediante una serie de señales básicas, siguiendo un orden topológico e indicando la conectividad (aristas) del grafo en forma de lista de incidencia (véase sección 1.4.1). Vamos a distinguir cuatro señales básicas:

- `vertex` (id): indica que hay un vértice activo y nuevo con identificador id.
- `is_initial` (p): indica que el vértice activo es un vértice inicial con probabilidad p de ser inicial.
- `change_stage`: esta señal se recibe cuando se produce un cambio de etapa en el algoritmo que genera el grafo de reordenamiento. El cambio de etapa nos permite conocer desde donde pueden llegar arcos a los nodos del grafo de entrada. Un nodo *nunca* puede recibir arcos de nodos que pertenecen a su misma etapa (ahora mismo no se está utilizando esta información, pero puede servir en el futuro para acelerar los cálculos del algoritmo).
- `edge` (id, data=$\{\langle r, p\rangle\}$): indica la existencia de una arista desde el vértice id. La arista *incide* en el vértice activo. Dicha arista está etiquetada con la posición r en la frase de entrada (palabra x_r), y con la probabilidad asociada a dicha arista en el grafo de entrada. Normalmente esta probabilidad será 1, ya que el módulo anterior es el módulo de reordenamiento que no genera probabilidades. No obstante, se podría utilizar otro tipo de generador de grafos de entrada que sí estimara probabilidades en las aristas.
- `no_more_in_edges` (id): indica que el vértice id (normalmente el vértice activo) no va a recibir más aristas.

Las estructuras de datos utilizadas son el transductor árbol de prefijos para generar tuplas, y una estructura interna que asocia a cada vértice posible en el grafo de entrada una lista de estados activos en dicho árbol de prefijos. Cada estado activo contiene:

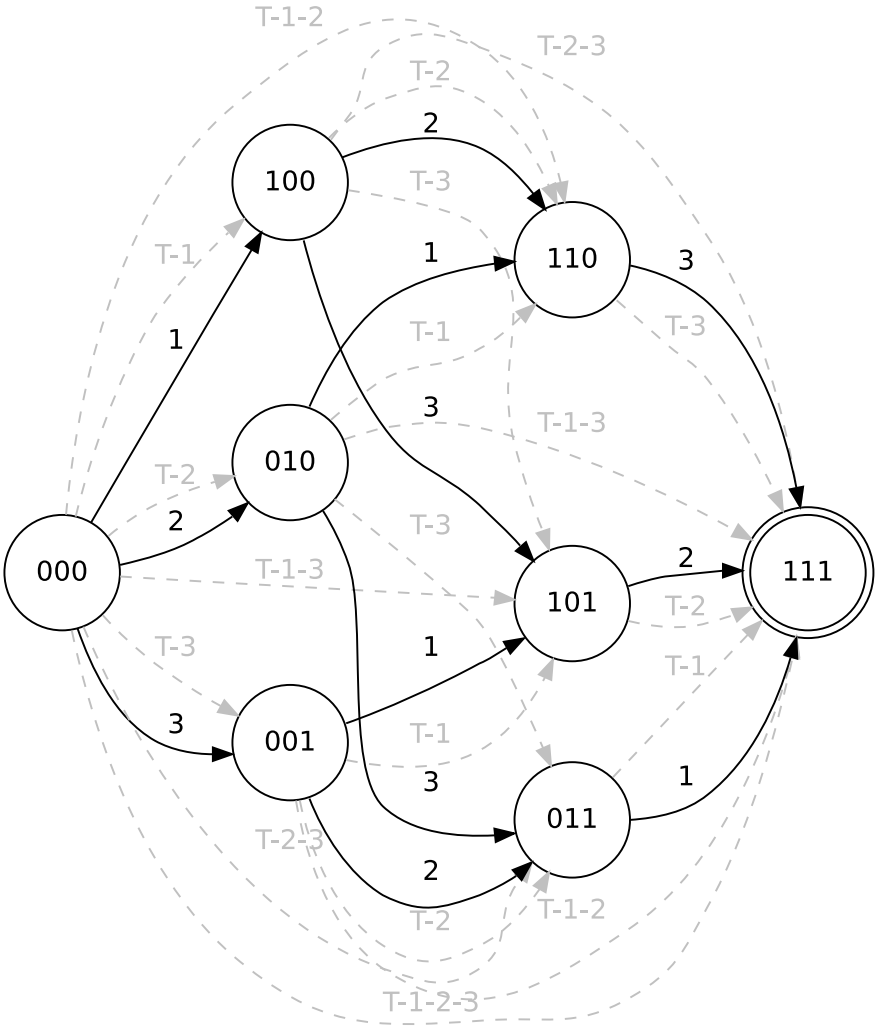


Figura 10.5: *Grafo donde se representa el reordenamiento de la frase de entrada, y en gris las aristas con tuplas añadidas por el módulo Word2Tuple. //* Word-reordering graph; tuple edges are represented in gray. Referenced at page 314.

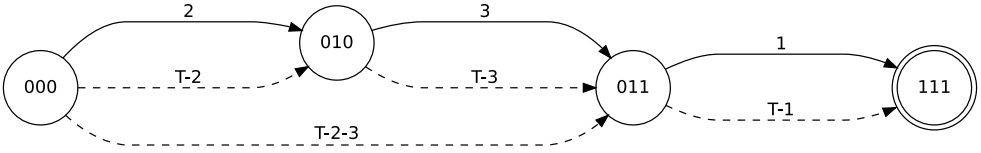


Figura 10.6: *Las aristas en color sólido son un camino en el grafo de reordenamiento, y las aristas discontinuas son las tuplas que el módulo Word2Tuple genera para dicho camino. //* Instance of a path in the tuple extended graph. Referenced at page 314.

- vértice origen en el grafo de entrada (clave del estado),
- estado activo en el árbol de prefijos (clave del estado),
- posición de la primera palabra cubierta por la hipótesis,
- posición de la última palabra cubierta por la hipótesis,
- puntuación de la hipótesis.

Se puede formalizar el algoritmo definiendo lo que hay que ejecutar con cada una de las acciones del protocolo de incidencia. El esquema seguido para la comunicación entre el proceso generador del grafo, Word2Tuple, y el algoritmo de Viterbi de la siguiente etapa, se basa en la sincronización mediante el método `score_feed_back` (p). Cuando Word2Tuple recibe la respuesta `score_feed_back` (p), esta es comunicada al objeto generador del grafo de reordenamiento (líneas 30 y 33) del algoritmo 10.5. El módulo Word2Tuple ejecuta la siguiente lógica con cada una de las señales que puede recibir:

- `vertex` (id): cuando llega esta señal se genera un vértice nuevo asociado al identificador id, con una lista de estados activos vacía, a la espera de recibir aristas para ir rellenando dicha lista, y marca dicho vértice como *vértice activo*. Por defecto la probabilidad de ser inicial es cero. Véase el algoritmo 10.3 para una descripción algorítmica de este procedimiento.
- `is_initial` (p): esta señal indica que la probabilidad de ser inicial del vértice activo es p.
- `edge` (id, data=$\{\langle r, p\rangle\}$): con cada arista se recorre la lista de estados activos asociada al vértice id. Para cada estado activo se comprueba si es posible transitar con la palabra x_r (palabra que ocupa la posición r en la frase de entrada). Si es posible, se transita, se aplica la probabilidad de la transición, y se genera en la lista de estados activos del vértice actual uno nuevo que tiene como vértice origen en el grafo de entrada el mismo que tenía el estado activo (no tiene porque ser id), como estado del árbol el resultante de la transición, y como puntuación la resultante de la transición. Se restringe la búsqueda forzando a que las posiciones de las palabras en la parte origen de las tuplas sea siempre monótonamente creciente, admitiendo saltos de más de una posición entre palabra y palabra. Esta condición puede ser más restrictiva todavía, obligando a que las palabras en la parte origen de la tupla sean siempre consecutivas. Restringir la búsqueda de este último modo permite que el algoritmo pueda trabajar con tablas de segmentos en lugar de con tuplas, y por tanto, el algoritmo puede funcionar usando la aproximación basada en segmentos y la aproximación basada en N-gramas. Véase el algoritmo 10.4 para una descripción algorítmica de este procedimiento.
- `no_more_in_edges` (id): cuando llega esta señal se recorre toda la lista de estados activos del vértice activo, comprobando si alguno genera tuplas en el transductor árbol de prefijos. Si es así, entonces se emite una arista que va desde el vértice origen del estado activo hasta el vértice actual, con la puntuación apropiada. En este momento se recibe el mensaje `score_feed_back` (p) que genera el módulo Viterbi, devolviendo la probabilidad del mejor estado activo en la búsqueda del proceso de traducción completo. Esta información es remitida hacía detrás al módulo de reordenamiento a través

de un nuevo mensaje `score_feed_back`. Esta información es muy importante, pues permite a la poda que implementan los módulos discernir mejores de peores hipótesis. Véase el algoritmo 10.5 para una descripción algorítmica de este procedimiento.

Este módulo genera la asociación dada por la función φ, y transmite la información de la posición en la frase de entrada de la primera y la última palabra de cada tupla. Dicha información es transmitida a través del mensaje `segment_info`.

Algoritmo 10.3 Word2Tuple: método `vertex`. // *Word2Tuple: method* `vertex`. *Referenced at page 314.*

Require: An input vertex id
Ensure: Generate an empty active states list associated to the input vertex id
1: **procedure** `vertex` (id)
2: **begin**
3: Let id the current active vertex
4: Let $V(id) = \{C = \emptyset, M_I = 1, p_I = 0\}$ an empty active states list with initial mass 1 and 0 probability of being initial vertex // p_I probability could be modified with a message `is_initial`
5: **end procedure**

10.4 Búsqueda del mejor camino en el grafo de tuplas: módulo Viterbi

Para buscar el mejor camino en el grafo de tuplas se utiliza una extensión del algoritmo de Viterbi para grafos visto en la sección 6.2.2. Vamos a comentar las diferencias entre ambos, sin reescribirlo entero.

Este módulo implementa el algoritmo de Viterbi y aplica la información de los modelos contextuales y de reordenamiento descritos en las secciones 9.5.2 y 9.5.3. En total hay cuatro modelos cuya puntuación se combina log-linealmente junto a la puntuación o probabilidad con la que Word2Tuple etiqueta las aristas. En el módulo de Viterbi original únicamente había un modelo, el modelo de lenguaje. Los modelos que combinamos en este módulo son:

- Uno o más modelos de lenguaje de tuplas.
- Uno o más modelos de lenguaje de la lengua destino.
- Modelo de reordenamiento simple.
- Modelo de reordenamiento lexicalizado.

Este módulo guarda listas de estados activos asociadas a los vértices del grafo que recibe a su entrada. Estas listas tienen un tamaño máximo, lo que permite aplicar poda estática basada en histograma. Cuando llega el mensaje `no_more_in_edges` se realiza la poda de las lista de estados asociada al vértice activo. Cada estado activo contiene:

Algoritmo 10.4 Word2Tuple: método `edge`. // *Word2Tuple: method* `edge`. *Referenced at page 315.*

Require: An input vertex id, a data list with only one (source-word-position,score) pair
Ensure: Compute a Viterbi step on all edges that income to the current active vertex

1: **procedure** `edge` (id, data=$\{\langle r, p\rangle\}$)
2: **begin**
3: **for all** active state key-value pair $((v_{id}, st) \rightarrow (b, e, p)) \in V(id).C$ **do**
4: Let first covered word position $b' = \begin{cases} b, & \text{iff } b \neq 0\,; \\ r, & \text{iff } b = 0\,; \end{cases}$
5: **if** $e < r$ (ensure monotonous increasing order, forward jumps are admisible) **then**
6: Let $e' = r$
7: Let w the word at position e' on source sentence $\bar{x}$
8: **if** $\exists$ transition $\delta(st, w)$ **then**
9: Let $st' = \delta(st, w)$ destination state
10: Let $p' = p \cdot P_T(st, w)$ transition probability
11: Let $p'' = 0$ a default probability value
12: **if** exists value associated to key (v_{id}, st') **then**
13: Let p'' the probability value of key $(v_{id}, st') \in V$(current active vertex).C
14: **end if**
15: **if** $p' > p''$ **then**
16: Set value of key $(v_{id}, st') \in V$(current active vertex).C to (b', e', p')
17: **end if**
18: **end if**
19: **end if**
20: **end for**
21: **end procedure**

Algoritmo 10.5 Word2Tuple: método `no_more_in_edges`. *// Word2Tuple module: method `no_more_in_edges`. Referenced at page 315.*

Require: An instance of parsing data structures
Ensure: Generate vertices and edges with bilingual tuples using active state list of the current active vertex

1: **procedure** `no_more_in_edges` ()
2: **begin**
3: **for all** key-value pairs $((v_{id}, st) \rightarrow (b, e, p)) \in V(\text{current active vertex}).C$ **do**
4: Let L an empty list of pairs $\langle z \in \Delta, probability \rangle$
5: **for all** possible tuple $z \in \Delta$ which $P_E(st, z) > 0$ **do**
6: **if** not generated message `vertex` for current active vertex **then**
7: Generate message `vertex` (current active vertex)
8: **if** $V(\text{current active vertex}).p_I > 0$ **then**
9: Generate message `is_initial` ($V(\text{current active vertex}).p_I$)
10: **end if**
11: **end if**
12: Compute output probability $p' = \dfrac{p \cdot P_E(st, z)}{V(\text{current active vertex}).M_I}$ // Output probability is the result of substracting the probability mass received from `score_feed_back` message
13: Append to L the pair $\langle z, p' \rangle$
14: **end for**
15: **if** L is not empty **then**
16: Generate message `segment_info` (b, e) to indicate the segmentation points of next edge
17: Generate message `edge` (v_{id}, data=L)
18: **end if**
19: **end for**
20: **if** message `vertex` was generated or $V(\text{current active vertex}).p_I > 0$ **then**
21: Generate message `vertex` if necessary
22: Generate message `is_initial` if necessary
23: Generate message `no_more_in_edges` (current active vertex)
24: Wait for message `score_feed_back` and store its result on p
25: Store initial mass $V(\text{current active vertex}).M_I = p$
26: **if** $V(\text{current active vertex}).p_I > 0$ **then**
27: Update $p = p \cdot V(\text{current active vertex}).p_I$
28: **end if**
29: Set value of key $(\text{current active vertex}, q_R) \in V(\text{current active vertex}).C$ to $(0, 0, p)$
30: Generate message `score_feed_back` (p) to previous dataflow module
31: **else**
32: Let $p = \max \{\text{probability value } p \in V[\text{current active vertex}].C\}$
33: Generate message `score_feed_back` (p) to previous dataflow module
34: **end if**
35: **end procedure**

- Estado en el modelo de lenguaje de tuplas. En caso de ser un modelo de N-gramas estándar, este estado se representa por un índice al autómata que guarda el modelo. En el caso de NN LMs, este estado es un LMkey al TrieLM (véase el capítulo 2). Aunque haya más de un modelo de lenguaje de tuplas, todos comparten el mismo camino, basta con tener un único identificador para todos.
- Estado en el modelo de lenguaje de N-gramas de la lengua destino. Igual que el anterior, en caso de ser un modelo de N-gramas estándar, este estado se representa por un índice al autómata que guarda el modelo. En el caso de NN LMs, este estado es un LMkey al TrieLM (véase el capítulo 2). Igual que antes, aunque tengamos más de un modelo de lenguaje de la lengua destino, basta con un único identificador para todos.
- Puntos de segmentación en la frase de entrada: guarda las posiciones de la **primera** palabra cubierta y la **última**, que corresponde a la arista con la que llegamos a este estado activo. Esta información es útil para los modelos de reordenamiento.
- Identificador de la tupla bilingüe que etiqueta la arista con la que llegamos hasta el estado activo actual.
- Probabilidad acumulada de llegar hasta dicho estado activo partiendo desde el estado activo inicial en el vértice inicial del grafo de palabras.
- Backpointer al estado activo desde el cual llegamos a este con dicha probabilidad.

Cada estado activo se encuentra en una lista asociada a cada vértice del grafo de entrada. Para cada vértice guardamos:

- Lista de estados activos.
- Probabilidad del mejor de los estados activos en la lista.
- Identificador del vértice.

Siguiendo con la arquitectura modular donde cada uno de ellos se comporta de forma reactiva a una serie de mensajes que pueden intercambiar entre ellos, el módulo Viterbi reacciona a los mismos mensajes que el módulo NgramParser de la sección 6.2.2, y a uno más, el mensaje `segment_info`, que indicará la segmentación en posiciones de la frase de entrada de la arista o grupo de aristas que van a llegar. El algoritmo 10.6 detalla la respuesta de este módulo al mensaje `edge`, ya que es donde más diferencias hay entre ambos módulos. Nótese que se ha simplificado mucho el algoritmo trabajando en un nivel de abstracción más elevado.

Al terminar el algoritmo, es posible solicitar la solución, al igual que al algoritmo de la sección 6.2.2, pidiendo la mejor de todas las hipótesis, la siguiente mejor hipótesis de forma incremental, o un grafo de palabras.

10.5 Ejemplo de ejecución del algoritmo

La figura 10.7 muestra un ejemplo del proceso de búsqueda completo, detallando la entrada y la salida de cada uno de los módulos. La figura 10.8 ilustra con detalle el último paso, la búsqueda mediante el algoritmo de Viterbi del camino de mayor probabilidad en el grafo de tuplas que recibe a su entrada. La figura representa todos los estados activos generados,

Algoritmo 10.6 Método `edge` del sistema de traducción automática estadística. // *`edge` method for the SMT Viterbi parser. Referenced at page 315.*

Require: Origin vertex id and tuple,score pair list with k hypothesis
Ensure: Apply contextual models probabilities and updates search active states

function `edge` (id, data=$\{\langle z_1, p_1\rangle, \langle z_2, p_2\rangle, \ldots, \langle z_k, p_k\rangle\}$)
begin
  for all active state $a \in$ Vertex(id) **do**
    for all pair $\langle z, p\rangle \in$ data **do**
      Let $p_{tlm}^{(i)} = \prod_{w_j \in t(z)} p(w_j|h_j)$ the probability of the $t(z)$ target words sequence with each of the i target language models computed using the language model interface of section 3.2 and the target language model LMkey stored at a
      Let $p_{tm}^{(j)} = p(z|h)$ the probability of bilingual tuple z with each of the j bilingual language models and the bilingual language model LMkey stored at a
      Let (b, e) the segmentation data stored at active state a
      Let (b', e') the segmentation data associated with current edge
      Let z' last transition tuple stored at active state a
      Let $p_{lr}^{(l)}$ the probability of each l lexicalized reordering model computed from parameters z, z', b, e, b', and e'
      Let p_d the probability of the word distortion model computed from parameters e and b'
      Let p'' the result of the log-linear combination of $p_{tlm}^{(i)}$, $p_{tm}^{(j)}$, $p_{lr}^{(l)}$, p_d, and p, using the corresponding coefficients λ, for each possible value of i, j, and l
      Let p' the probability value stored at active state a
      Update $p' = p' \cdot p''$
      Let k' the active state key corresponding to target language model LMkey, the bilingual language model LMkey, segmentation point (a', b'), and tuple z
      Let a' the new active state, with probability p' and a backpointer to a
      if p' outperforms previous value associated with k' at Vertex(current active vertex) **then**
        Update active state associated with k' at Vertex(active state vertex) using a' information
      end if
    end for
  end for
end function

agrupados por el vértice del grafo de entrada al que están asociados. En negrita se han marcado los punteros hacia detrás que forman el mejor camino. En esta figura cada estado activo muestra la siguiente información:

- TLM LMkey: es el identificador del estado del modelo de lenguaje de la lengua destino.
- BLM LMkey: es el identificador del estado del modelo de lenguaje bilingüe $\mathcal{ST}'$.
- Last tuple: última tupla con la que se transitó.
- Seg. info.: información sobre la posición en la frase de entrada de la primera y la última palabra de la última tupla con la que se transitó.
- Backpointer: puntero hacia detrás al estado activo desde el cual llegamos.

En ambas figuras las probabilidades han sido eliminadas de los arcos por claridad. Las figuras no permiten conocer el orden en que se producen los grafos, y como fluye la información de uno a otro módulo. Es necesario revisar la descripción que hace este capítulo de los algoritmos para entender bien este punto. Todos los algoritmos generan un grafo y la comunicación de dicho grafo es sincronizada a través de los mensajes `no_more_in_edges` y `score_feed_back`.

10.6 Parámetros de configuración del sistema de traducción

Los algoritmos descritos en este capítulo forman una arquitectura de sistema de traducción configurable a diferentes niveles. Esta sección va a especificar dichos parámetros para acercar al lector a los detalles del sistema de traducción desarrollado.

Parámetros de los modelos incontextuales

- Longitud máxima de los segmentos: establece la longitud máxima de los segmentos extraídos. Son eliminadas aquellas tuplas en las que cualquiera de sus dos segmentos supera esta longitud (por defecto fijada a 7).
- Vocabularios: el sistema recibe tres vocabularios, el vocabulario de la lengua de entrada Σ, el vocabulario de la lengua destino Ω y el vocabulario de tuplas bilingües o pares de segmentos Δ.
- Tabla de tuplas o tabla de segmentos: una tabla donde se indica la puntuación de los modelos incontextuales para cada tupla o par de segmentos. Nótese que esta tabla no incorpora ni el WIP ni el TIP.
- Un valor para el parámetro WIP y otro para el TIP.
- Valor de la penalización por cada palabra desconocida (por defecto -100 en escala logarítmica).

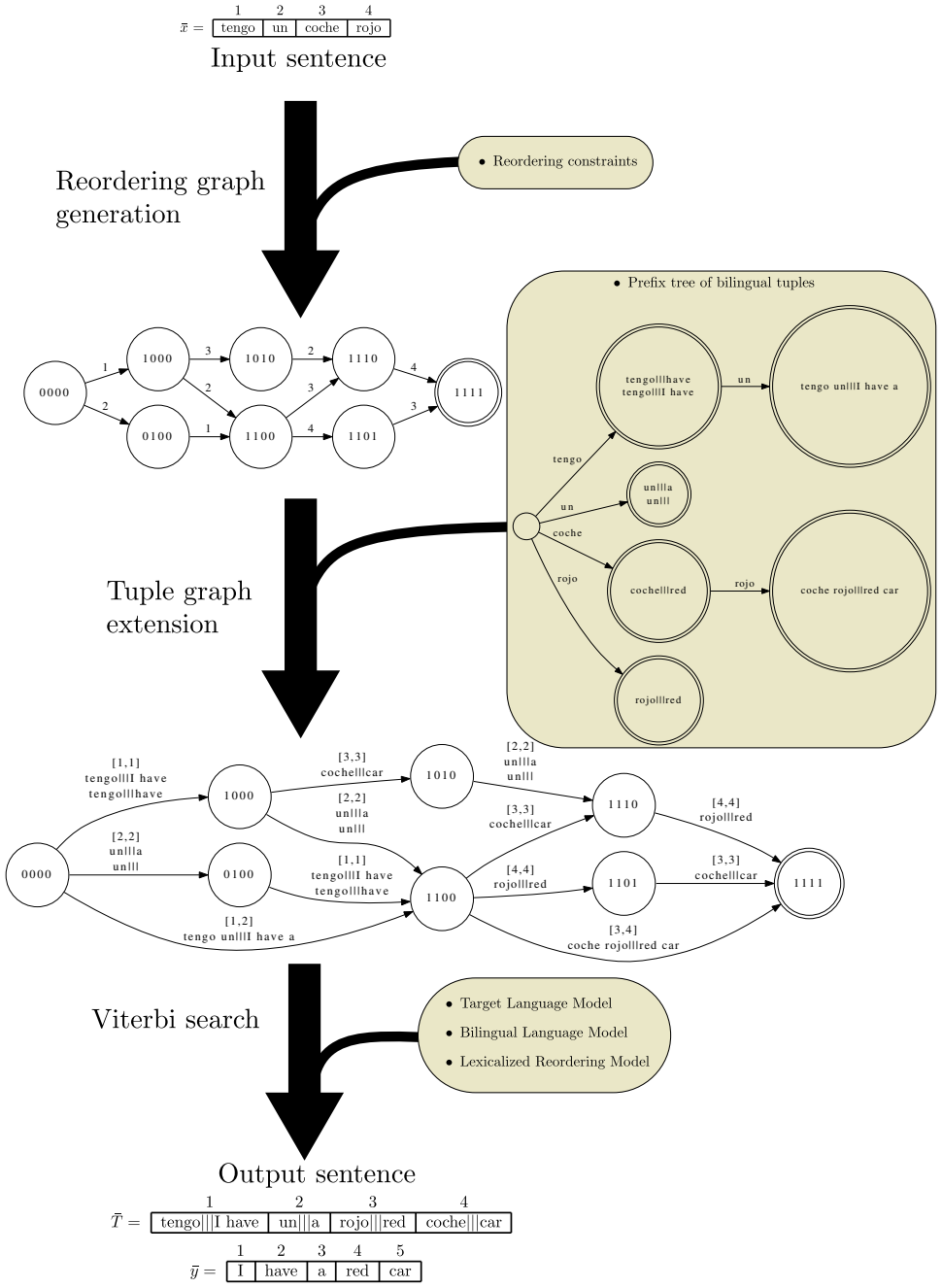


Figura 10.7: *Ejemplo de ejecución del algoritmo de búsqueda tipo dos pasos desarrollado para tareas de traducción automática estadística. La frase de entrada es* tengo un coche rojo*. Se muestra la entrada y la salida producida por cada uno de los módulos del algoritmo. La figura 10.8 detalla el trellis del algoritmo de Viterbi. //* Instance of the SMT decoding search execution. The input sentence is "tengo un coche rojo". Referenced at page 315.

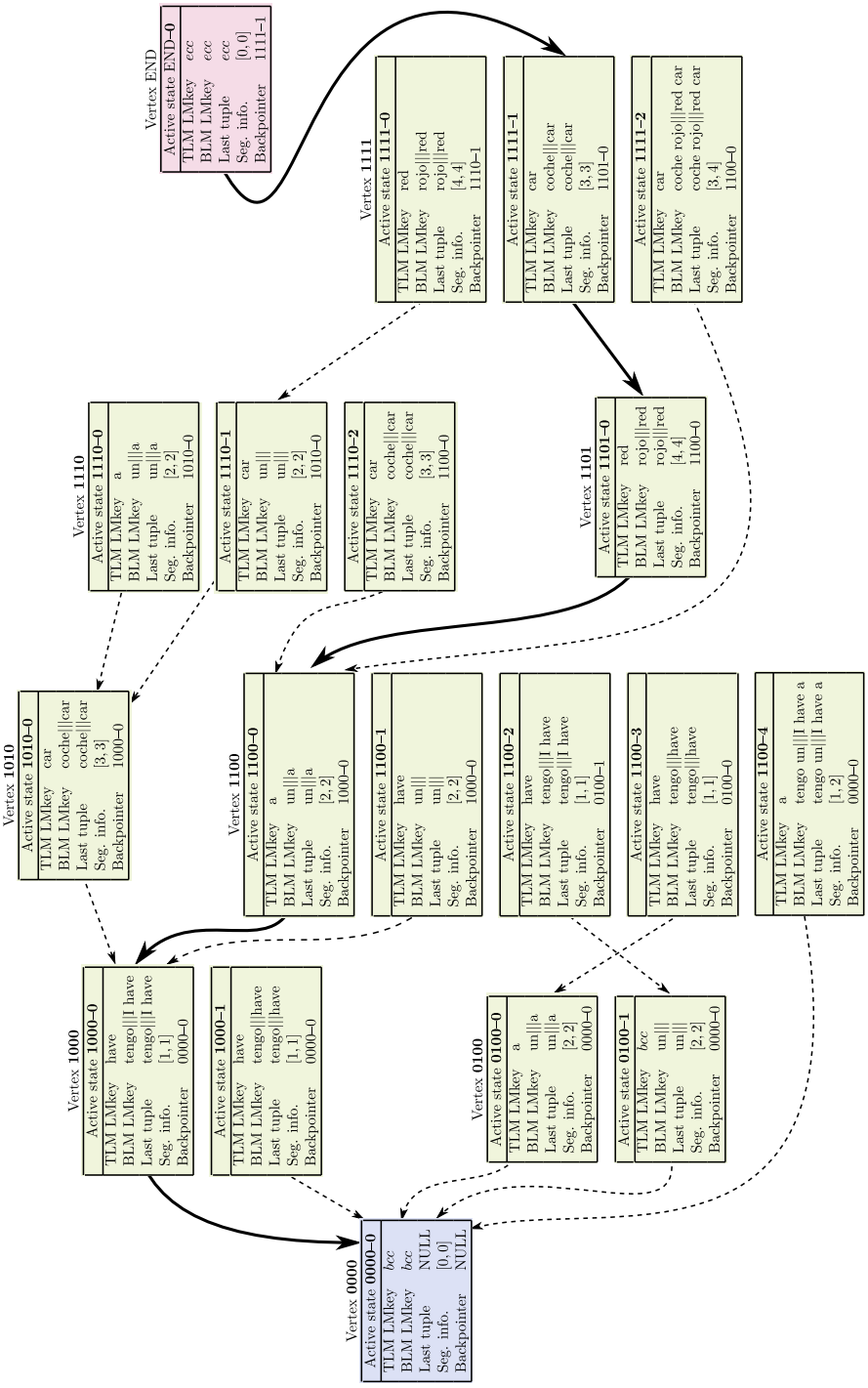


Figura 10.8: *Ejemplo de trellis del algoritmo de Viterbi para la instancia de entrada de la figura 10.7 y modelos de lenguaje de bigramas. Los arcos son backpointers y el mejor camino está representado en negrita. La sección 10.5 da más detalles de la figura. //* Trellis generated by the SMT Viterbi module using bigrams. Edges are backpointers. The best path is bolded. Referenced at page 315.

Parámetros del módulo de reordenamiento

- Tamaño j de la ventana de reordenamiento: configura el tamaño de la ventana usada para generar el grafo de reordenamiento (por defecto 5). El número de nodos del grafo de reordenamiento depende de este valor j y del tipo de reordenamiento. Para reordenamiento de tipo *local* el máximo posible es $|\bar{x}| \cdot 2^j$.

- Tamaño m de la poda en la etapa de reordenamiento: indica el número de nodos expandidos que van a sobrevivir en cada etapa de generación del grafo de reordenamiento (por defecto 10). Si el valor de m es mayor o igual que el máximo de nodos que se puede generar en cada etapa del algoritmo, entonces m no tiene efecto alguno.

- Tipo r de reordenamiento: puede ser 'B' para reordenamiento IBM, 'i' para reordenamiento "inverse IBM" o 'L' para reordenamiento local (por defecto 'L').

- Valores inicial y final para la búsqueda en haz (por defecto no se utilizan).

- Coste futuro: indica si utilizar o no el coste futuro para mejorar la poda (por defecto activado).

- Paredes: un vector con aquellas palabras que hacen la función de pared (por defecto `. , : ; - -- ? !`).

Parámetros del módulo Word2Tuple

- Poda de las tuplas o tabla de segmentos: permite restringir el número de tuplas máximo que pueden emitirse en un nodo del árbol de prefijos $\mathcal{A}$. En cada nodo de dicho árbol tenemos todas las tuplas (o pares de segmentos) que comparten la misma secuencia de palabras en la entrada. La poda se hace quedándose con las n que tienen mayor puntuación en la combinación log-lineal de los modelos incontextuales (por defecto 20).

- Búsqueda estrictamente monótona o búsqueda creciente: permite adaptar el algoritmo para modelos basados en tabla de segmentos o modelos basados en tuplas (por defecto búsqueda creciente).

Parámetros del módulo Viterbi

- Valor de la poda por histograma: establece el número de estados activos que deben sobrevivir cuando llega el mensaje `no_more_in_edges` (por defecto 50).

- Modelo de reordenamiento lexicalizado: en caso de establecerse este parámetro, indica donde está la tabla con los parámetros del modelo.

- Modelo de lenguaje de la lengua destino: permite una lista de modelos de lenguaje. Todos ellos serán combinados log-linealmente.

- Modelo bilingüe de lenguaje: admite una lista de modelos bilingües de lenguaje. Esta lista estará vacía si se quiere ejecutar para modelos basados en segmentos.

Tabla 10.1: *Resultados en BLEU y TER para cuatro configuraciones del sistema de traducción en las dos direcciones de la traducción de Inglés-Español. A1) Sistema baseline sin coste futuro. B1) sistema A1 con modelo de reordenamiento lexicalizado. A2) sistema A1 usando coste futuro para poder el grafo de reordenamiento. B2) sistema A2 con modelo de reordenamiento lexicalizado. //* BLEU and TER using four configurations of the SMT decoder. A1) Baseline system without future cost. B1) A1 system with lexicalized reordering models. A2) A1 system with future cost. B2) A2 system adding lexicalized reordering models.

	News2008		News2009	
System	BLEU	TER	BLEU	TER
Spanish–English				
Moses	19.6	63.8	20.4	60.3
A1	18.0	65.2	18.4	61.7
B1	18.1	65.0	18.7	61.7
A2	19.2	64.0	20.0	60.5
B2	19.8	63.8	20.2	60.4
English–Spanish				
Moses	20.0	64.3	20.3	62.5
A1	18.3	65.9	18.4	64.2
B1	19.0	65.5	19.2	63.7
A2	19.1	65.0	19.5	63.0
B2	19.8	64.5	20.0	62.7

10.7 Evaluando el sistema de traducción completo

10.7.1. Heurístico para el coste futuro y modelos de reordenamiento

Para evaluar el sistema se ha entrenado el modelo de traducción y el modelo de lenguaje mediante el corpus News-Commentary 2010 (véanse las tablas 11.4 y 11.5 del siguiente capítulo). Para la optimización de los pesos se ha utilizado el conjunto News2008 del WMT, y test News2009 del WMT. Se han realizado tres experimentos (véase tabla 10.1):

A) Sistema baseline usando 4-gramas para el modelo bilingüe de traducción y el modelo de lenguaje destino. $P = 40$ y la longitud máxima de la parte origen o destino de las tuplas es 7. El sistema de traducción utiliza las probabilidades léxicas tipo IBM-1 directa e inversa, las probabilidades condicionales directa e inversa, penalización del número de tuplas y del número de palabras en la lengua destino. El coeficiente de penalización de número de palabras desconocidas se fija durante el MERT a -100. El sistema utiliza la aproximación (a) para solucionar el problema de las palabras empotradas.

B) Añade al sistema anterior el modelo bidireccional de reordenamiento lexicalizado o "lexicalized bidirectional reordering".

De estos dos sistemas se han probado dos configuraciones, sin usar coste futuro y usándolo durante la poda (A1 y A2, B1 y B2) del módulo de reordenamiento.

Se puede observar como el uso del modelo de reordenamiento lexicalizado permite que el sistema mejore sus resultados, tanto en BLEU como en TER. Se destaca también el efecto

positivo del uso del coste futuro durante la primera etapa del sistema, cuando se genera el grafo de reordenamiento. En este punto la poda de un estado significa no seguir explorando aquellos caminos que comparten las misma palabras, aunque con distinto orden, en la lengua origen. Es una poda muy dura, y tener información sobre el coste futuro, aunque no sea perfecta, permite mejorar este paso de una forma muy significativa.

A modo comparativo se han puesto en la tabla resultados con Moses utilizando exactamente el mismo corpus y los mismos conjuntos de desarrollo y test. Se puede observar que las diferencias en ambas direcciones no son significativas, si bien Moses alcanza resultados mejores.

10.7.2. Resultados comparativos entre Moses y el sistema desarrollado

Para comparar el sistema implementado en `April` y la herramienta Moses se han entrenado modelos de traducción utilizando el corpus News-Commentary 2010 concatenado con el corpus Europarl v5. Se han realizado experimentos con diferentes pares de lenguas en una única dirección, Español–Inglés (véanse las tablas 11.4 y 11.5 del siguiente capítulo) y Alemán–Inglés. La optimización de pesos se ha realizado con el conjunto News2008 del WMT, como test interno se ha utilizado el conjunto News2009 y para test el conjunto News2010. Para el modelo de lenguaje se ha utilizado la parte monolingüe correspondiente a Europarl v5 y News-Commentary 2010. La tabla 10.2 resume los resultados obtenidos. Se han probado los siguientes sistemas:

- Moses: es la configuración por defecto de Moses utilizando modelos de reordenamiento lexicalizados.
- `April` NGram-based: es el algoritmo de traducción automática basada en N-gramas descrito en este capítulo. La configuración incorpora todas las características y modelos que lo hacen comparable a Moses. No incorpora NN LMs.
- `April`* Phrase-based: de nuevo es el algoritmo de traducción descrito en este capítulo pero, en lugar de configurarse para utilizar los modelos de traducción basada en N-gramas, se ha configurado para utilizar la tabla de segmentos de Moses y los modelos de reordenamiento de Moses. Para poder llevar a cabo la decodificación hay que restringir la búsqueda en el método `edge` del módulo Word2Tuple obligando la secuencialidad de la parte origen de los segmentos.
- Moses*: a modo comparativo este sistema traduce utilizando Moses pero usando los pesos de la combinación log-lineal estimados por `April`*.

Estos resultados validan el sistema de traducción desarrollado, siendo competitivo con el estado del arte en traducción automática. Comparando la aproximación basada en N-gramas con Moses observamos que hay pequeñas diferencias, ninguna de ellas significativas. Si comparamos el sistema `April`* con Moses* observamos el resultado también es *casi* idéntico.

10.8 Algunos apuntes para mejorar la eficiencia computacional del algoritmo

Cabe destacar que la implementación actual del algoritmo de búsqueda incorpora memorias caché a dos niveles en los NN LMs (véase sección 4.3), esta es una forma de evitar hacer

Tabla 10.2: *Resultados en BLEU y TER comparando los resultados de traducción de Moses con la herramienta desarrollada para esta tesis (`April`). `April`⋆ es el sistema `April` pero utilizando modelos Phrase-based en lugar de N-grama-based. Moses⋆ es la herramienta Moses utilizando los pesos de la combinación log-lineal estimados para `April`⋆.* // BLEU and TER for Moses and `April` decoders. `April`⋆ is configured for phrase-based decoding. Moses⋆ uses Moses with log-linear weights computed with `April`⋆.

	News2008		News2009		News2010	
System	BLEU	TER	BLEU	TER	BLEU	TER
Spanish–English						
`April` N-gram-based	22.4	61.3	23.6	57.6	26.2	54.5
Moses	22.3	61.3	23.7	57.4	25.9	54.7
`April`⋆ Phrase-based	22.4	61.4	23.8	57.4	26.3	54.5
Moses⋆	–	–	–	–	26.2	54.6
German–English						
`April` N-gram-based	19.3	65.0	18.4	64.2	19.7	63.1
Moses	19.0	65.5	18.3	64.5	19.5	63.2
`April`⋆ Phrase-based	19.1	65.3	18.3	64.5	19.5	63.2
Moses⋆	–	–	–	–	19.5	63.0

cálculos dos veces, y puede mejorarse en diversos sentidos. Por ejemplo, se puede modificar la caché para que los eventos que aparecen muchas veces no se pierdan.

El algoritmo implementado ha sido diseñado para poder sincronizar la ejecución de la búsqueda a través del mensaje `change_stage`, que indica a través del protocolo de incidencia para grafos que se ha producido un cambio de etapa en el algoritmo. Todos los nodos del grafo que son generados entre dos mensajes `change_stage` nunca pueden estar conectados entre sí. Esto permite guardar todas las aristas que llegan a los nodos de una misma etapa, y procesarlas todas a la vez. De esta manera podemos implementar el modo bunch en en el forward del NN LM, y al mismo tiempo podemos aprovechar aquellas aristas que comparten transiciones en el modelo de lenguaje.

No obstante esta modificación de la implementación del algoritmo implica cambios importantes respecto a la implementación original, que fue ideada para problemas de reconocimiento de secuencias, sin tener en cuenta las particularidades de la traducción automática. A pesar de estos cambios, la esencia del algoritmo es la misma que se ha descrito en este capítulo.

Dejamos como trabajo futuro la implementación del algoritmo para que pueda aprovechar al máximo la eficiencia que nos permite el mensaje `change_stage`.

10.9 Resumen

Este capítulo describe los algoritmos desarrollados para poder llevar a cabo los experimentos en tareas de traducción automática estadística. El objetivo ha sido:

- mostrar la generalidad de los algoritmos desarrollados para reconocimiento de secuencias, que han podido adaptarse para funcionar en traducción automática;

- permitir la incorporación de NN LMs en el proceso de traducción de forma totalmente acoplada.

Los algoritmos se han descrito de forma reactiva, de manera que cada uno de ellos ejecuta un procedimiento en función de un mensaje que le llega. Esto permite que los algoritmos sean descritos dentro de una arquitectura dataflow donde cada proceso de la búsqueda es ejecutado por un módulo y es constituido por varios algoritmos. Estos módulos envían mensajes a otros módulos sincronizando el proceso de búsqueda haciendo que a todos los efectos sea equivalente a una búsqueda donde todo el sistema es acoplado.

Adicionalmente, se ha comparado el algoritmo de traducción aquí descrito con la herramienta Moses, cuyos resultados son estado del arte en traducción automática. Las diferencias entre ambos sistemas no son significativas, lo que permite afirmar que el sistema de traducción `April` es *competitivo* con el estado del arte.

11

EXPERIMENTACIÓN EN TRADUCCIÓN AUTOMÁTICA

Índice

Ya hemos explicado los detalles más importantes de la modelización llevada a cabo para traducción automática estadística (SMT), así como el algoritmo de decodificación implementado en el sistema. Este capítulo expone los resultados obtenidos a través de la combinación de herramientas estándar para SMT (como Moses y SRILM) con los modelos conexionistas

de lenguaje, así como los resultados obtenidos utilizando el sistema de traducción automática basada en N-gramas que integra en su interior los NN LMs.

Esta experimentación permitirá validar el coste computacional de la integración de los NN LMs en tareas de SMT, así como observar si hay alguna diferencia en la calidad del sistema. Al mismo tiempo servirá para validar la implementación del algoritmo de decodificación expuesto en el capítulo anterior.

11.1 Preproceso de datos en SMT

Antes de empezar con la descripción de la experimentación, es importante poner en relieve el preproceso de los datos que se ha seguido para estimar los modelos de traducción y de lenguaje:

- Proceso de *troceado* ("tokenization" en inglés) de las palabras del corpus original por separado para ambas lenguas. Para este cometido se utiliza la herramienta `tokenizer.perl` ofrecida por el toolkit Moses [Koehn et al., 2007]. Este procedimiento consiste en separar signos de puntuación de las palabras a las que están pegados y separar contracciones si las hubiera, sin modificar los acrónimos.
- Se pasa a minúsculas todo el texto.
- Se eliminan del corpus aquellas frases con más de 40 palabras en cualquiera de las lenguas. Esto permite reducir el coste computacional del alineamiento entre palabras, pese a menguar la cantidad de material disponible para entrenar el modelo de traducción. Dado que para entrenar el modelo de lenguaje este paso no es necesario, y por tanto no se realiza, cabe destacar que las tablas con estadísticas del corpus bilingüe y del corpus para estimar el modelo de lenguaje de la lengua destino no son idénticas.

Esto significa que, una vez traducido el texto, podemos medir el error sobre un conjunto de datos tras aplicarles este preproceso, o bien podemos revertir los cambios y medir el error sobre las frases originales. Cada sección experimental de este capítulo indica el caso que se aplica.

La figura 11.1 ilustra un ejemplo de frase antes y después de preprocesarla.

```
Mr President, concerning item 11 of the Minutes  on the order
of  business,  we  agreed  yesterday  to  have the Bourlanges
report on today's agenda.

mr president , concerning item 11 of the minutes on the order
of  business  , we  agreed  yesterday  to have the bourlanges
report on today 's agenda .
```

Figura 11.1: *Ejemplo de frase antes y después de preprocesarla para SMT. //* Instance of the preprocessing step for SMT.

11.2 Experimentos repuntuando listas de N-best

En este ámbito se han desarrollado diversos trabajos, algunos de ellos en colaboración con otros grupos de investigación, utilizando los modelos conexionistas de lenguaje en una etapa de rescoring de listas de N-best. Comentaremos en detalle los más relevantes para el desarrollo de la tesis:

- Dos trabajos preliminares con los pares de lenguas Italiano-Inglés [Khalilov *et al.*, 2008b] y Árabe-Inglés [Khalilov *et al.*, 2008a] donde el sistema de traducción está basado en N-gramas, utilizando la implementación de la herramienta Marie [Crego *et al.*,

Tabla 11.1: *Estadísticas de los conjuntos de entrenamiento, validación y test del corpus Italiano-Inglés del IWSLT'06. //* Statistics of the Italian-English corpus from the IWSLT'06 task.

Set	Language	Sentences	Words	Voc. size	References
Train	Italian	24.5K	166.3K	10.2K	–
Train	English	24.5K	155.4K	7.3K	–
Dev	Italian	489	5.2K	1.2K	7
Dev	English	489	5.6K	1.7K	7
Test	Italian	500	6.0K	7.3K	7
Test	English	500	7.3K	2.3K	7

2005b,a]. Estos dos trabajos se han ampliado y mejorado para enviar una comunicación a algún congreso del campo [Khalilov *et al.*, 2012].

- Dos participaciones en el "Workshop Of Machine Translation" (WMT), los años 2010 [Zamora-Martínez & Sanchis, 2010] y 2011 [Zamora-Martinez & Castro-Bleda, 2011], para el par de lenguas Inglés-Español, utilizando el sistema de traducción basada en segmentos Moses [Koehn et al., 2007]. En el apéndice C se pueden encontrar las tablas C.1 y C.3 con los resultados oficiales de ambas evaluaciones.

11.2.1. Tarea Italiano-Inglés del IWSLT'06

Vamos a exponer una extensión y mejora de los resultados publicados en [Khalilov *et al.*, 2008b]. Con esta experimentación se pretende medir los resultados obtenidos repuntuando listas de N-best mediante NN LMs utilizando la tarea del par de lenguas Italiano-Inglés perteneciente al IWSLT'06. Se probaron diferentes configuraciones de vocabulario a la entrada y a la salida del NN LM. El criterio de selección del vocabulario fue la frecuencia absoluta de aparición de cada palabra, quedándonos con aquellas que aparecen más de Θ veces. En la tabla 11.1 se muestran las estadísticas del corpus utilizado, y en la tabla 11.2 el tamaño del vocabulario utilizado para los NN LMs para valores de $\Theta = 2, 3, 4$.

Los datos han sido preprocesados para homogeneizar el corpus. Se ha procedido a trocear, etiquetar, lematizar y separar las contracciones de la parte italiana siguiendo la descripción de [Crego *et al.*, 2006a].

Para ofrecer una comparación más justa, se ha considerado el uso de un modelo de lenguaje de N-gramas estándar para hacer rescoring de listas de N-best del mismo modo que se hace con los NN LMs. Este será el sistema denominado *Baseline*. Como configuración alternativa se ha considerado la inclusión del modelo de lenguaje de N-gramas estándar como un modelo más de la combinación log-lineal durante la decodificación. Los resultados de este sistema son los correspondientes a un sistema estándar de traducción automática basada en N-gramas.

Entrenamiento de los modelos de lenguaje

Los modelos de N-gramas estándar se han estimado con el descuento de Kneser-Ney modificado y con un valor de $N = 4$ (4-gramas). Nótese que al tratarse de traducción basada en

Tabla 11.2: *Número de palabras en el vocabulario de la Short-List, tanto a la entrada como a la salida del NN LM utilizado en la tarea Italiano-Inglés del IWSLT'06, contando únicamente aquellas palabras cuya frecuencia absoluta de aparición sea mayor que un umbral* Θ*. El conjunto de entrenamiento tiene un total de* 10.2*K palabras de vocabulario. //* Short-List sizes taking into account words whose frequency is greater than Θ. The full vocabulary has 10.2K words.

Frequency $> \Theta$	Short-List: # of vocabulary words
4	2 498
3	3 093
2	4 105

N-gramas, el modelo de traducción es un modelo de lenguaje bilingüe. Por tanto tendremos dos modelos de 4-gramas, el de la lengua destino (Inglés en este caso) y el de traducción.

Se utiliza la aproximación comentada en el capítulo 3 para solucionar el problema de la Short-List de los NN LMs, añadiendo al vocabulario de la Short-List una palabra más que sirve para estimar la probabilidad de todas las palabras fuera de la Short-List (OOS). Esta masa de probabilidad se reparte entre todas esas palabras de forma proporcional a su frecuencia relativa de aparición en el conjunto de entrenamiento. Los NN LMs son el resultado de combinar linealmente 4 redes neuronales, cada una con una capa de proyección diferente, tomando los valores $P = 128, 160, 192, 224$. La capa oculta es de 200 neuronas en todos los casos. Se han entrenado 3-gramas y 4-gramas.

Los pesos λ de la combinación log-lineal de la ecuación 1.3 se estiman utilizando como función objetivo *100 BLEU + 4 NIST*. Esta es una de las configuraciones posibles con Marie. El MERT utiliza el algoritmo de optimización del Simplex [Nelder & Mead, 1965].

Resultados y conclusiones

La tabla 11.3 muestra los resultados en BLEU para los sistemas baseline y los sistemas que utilizan NN LMs, tanto para los conjuntos de validación como para el test, con distintos tamaños de la Short-List.

Como se puede observar, se ha logrado una mejora de algo más de un punto de BLEU. En todos los casos los resultados al utilizar NN LMs *mejoran* los sistemas baseline. El mejor sistema ha sido el 4-grama NN LM con $\Theta = 2$, que obtiene una mejora de 1.2 puntos de BLEU en el test si lo comparamos con el sistema N-grama, y de 1.3 puntos de BLEU si comparamos con el sistema baseline.

Todas las diferencias son estadísticamente significativas para un intervalo de confianza del $95\,\%$ y usando el método pairwise comparison descrito en la sección 1.6. La cota superior del BLEU para el sistema Baseline es de 34.0 puntos de BLEU.

11.2.2. Tarea Inglés-Español del WMT'10

Los experimentos de esta sección son el resultado de la participación en el "Workshop of Machine Translation" (WMT) del año 2010. EL WMT es una cita anual de la comunidad científica de traducción automática en la que los distintos grupos de investigación proponen sus ideas y estas son evaluadas usando los mismos recursos. El WMT sirve como punto de referencia para comparar sistemas de traducción de forma justa.

Tabla 11.3: *Resultados en BLEU con el conjunto de validación y el conjunto de test para la tarea Italiano-Inglés del IWSLT'06. //* BLEU for development and test sets from the Italian-English IWSLT'06 task.

System		Dev	Test
Baseline (with LM rescoring)		39.2	33.5
N-gram (with LM integrated)		39.4	33.6
NN LMs for rescoring			
NN LM $\Theta = 4$	3-gram	41.7	34.3
	4-gram	40.9	34.4
NN LM $\Theta = 3$	3-gram	41.6	34.4
	4-gram	42.0	34.5
NN LM $\Theta = 2$	3-gram	41.8	34.2
	4-gram	42.3	34.8

Tabla 11.4: *Estadísticas del corpus bilingüe Inglés-Español del WMT'10 extraídas del corpus limpiado, troceado y pasado a minúsculas. News2008 se usa como conjunto de desarrollo y News2010 como test final. //* English-Spanish WMT'10 corpus statistics, after cleaning, tokenization and lowercasing. News-2008 is the development set, and News2010 is an internal test set.

	English		Spanish	
Set	# Lines	# Words	# Lines	# Words
News-Commentary 2010	80.9K	1.6M	80.9K	1.8M
Europarl v5	1.3M	26.6M	1.3M	27.5M
United Nations	4.3M	85.8M	4.3M	95.8M
News2008	2.0K	49.7K	2.0K	52.6K
News2010	2.5K	61.9K	2.5K	65.5K

Para el WMT'10 se preparó un sistema de traducción utilizando Moses [Koehn et al., 2007]. Se hicieron experimentos con diferentes tamaños de la lista de N-best. La tabla 11.4 resume las estadísticas de la parte bilingüe del corpus del WMT'10, mientras que la tabla 11.5 muestra las estadísticas más relevantes de la parte Española del corpus utilizado. El modelo de lenguaje estándar fue entrenado utilizando toda esta información, mientras que el conexionista únicamente con la parte relativa a News.

Entrenamiento del modelo de lenguaje

Se entrenó un 6-grama NN LM, formado por una capa de proyección de 640 neuronas (128 por cada palabra), una capa oculta de 500 neuronas y una capa de salida de 20K neuronas. Esto implica que se utilizó una Short-List con las 20K palabras más frecuentes, tanto en la entrada como en la salida del modelo.

El NN LM fue entrenado utilizando como datos de entrenamiento y validación dos particiones aleatorias extraídas de la concatenación de News y News-Commentary 2010. Para

Tabla 11.5: *Estadísticas del corpus Español de la tarea Inglés-Español del WMT'10, calculadas tras trocear y pasar a minúsculas. //* WMT'10 Spanish corpus statistics, after tokenization and lowercasing.

Set	# Lines	# Words	Voc. size
News-Commentary 2010	108K	2.96M	67K
News-Shuffled	3.86M	107M	512K
Europarl	1.82M	51M	172K
United Nations	6.22M	214M	411K
Total	3.96M	110M	521K

Tabla 11.6: *Calidad de la traducción Inglés-Español de la tarea WMT'10 medida en BLEU/TER para el conjunto de desarrollo y el test oficial. //* BLEU/TER for the WMT'10 English-Spanish task.

	News2008 (dev)		News2010 (test)	
System	BLEU	TER	BLEU	TER
Baseline	24.8	60.0	26.7	55.1
NN LM	25.2	59.6	27.8	54.0

entrenamiento se usaron 3M frases (102M palabras), y para validación se usaron 300K frases (8M palabras). En cada iteración del algoritmo de entrenamiento se utilizó un valor de reemplazamiento de 300K. La red necesitó 129 épocas para lograr la convergencia, lo que supone que únicamente fue entrenada con 38.7M palabras del conjunto de entrenamiento, escogidas aleatoriamente y con reemplazamiento. La perplejidad obtenida por el modelo fue de 116 puntos en el conjunto de desarrollo News08, frente a los 94 que logra el modelo de lenguaje de 6-gramas estándar estimado mediante el suavizado Kneser-Ney [Kneser & Ney, 1995].

Resultados cambiando el tamaño de la lista de N-best

El número de frases en la lista de N-best fue fijado a las 1 000 mejores hipótesis *únicas*. Los resultados se pueden ver en la tabla 11.6. Los resultados mostrados en la tabla mejoran el baseline de forma estadísticamente significativa utilizando el test pairwise comparison de [Koehn, 2004].

Para ilustrar el efecto del tamaño de la lista de N-best, se han efectuado experimentos reduciendo el tamaño de la misma. Los resultados se muestran en la tabla 11.7.

Conclusiones

En esta campaña se presentó un sistema que incorporaba como característica extra el uso de NN LMs para hacer rescoring de las listas de N-best generados con Moses. El análisis de los resultados en función del tamaño de las listas de N-best muestra que dicho tamaño está correlado con la calidad de los resultados. El sistema final obtuvo una gran mejora respecto al baseline, 1.1 puntos absolutos de BLEU y 1.1 puntos absolutos de WER, como se puede ver en la tabla 11.6. Estos resultados indican que el efecto de los NN LMs es *muy significativo*, incluso cuando han sido entrenados sólo con datos del dominio de la tarea (en este caso

Tabla 11.7: *Calidad de los resultados en BLEU/TER para diferentes tamaños de lista de N-best con el conjunto oficial de test de la tarea Inglés-Español del WMT'10. //* BLEU/TER using different N-best-list sizes for the official test set from the English-Spanish WMT'10 task.

N-best list size	BLEU	TER
200	27.5	54.2
400	27.6	54.2
600	27.7	54.1
800	27.6	54.2
1000	27.8	54.0

News). El sistema quedó en sexta posición en la tabla oficial de resultados de la organización del WMT'10 (véase apéndice C, tabla C.1).

11.2.3. Tarea Inglés-Español del WMT'11

En esta campaña volvió a utilizarse un sistema de traducción basado en Moses. En adición al modelo conexionista de lenguaje se diseñaron experimentos para mejorar la estimación de modelos de traducción cuando se utilizan grandes cantidades de datos fuera del dominio de la tarea [Matsoukas *et al.*, 2009; Foster *et al.*, 2010]. En este caso, la tarea es de noticias, y el corpus utilizado para entrenar el modelo de traducción incorpora actas del parlamento europeo, actas de las naciones unidas y una pequeña parte de noticias.

En la línea de mejorar el modelo de traducción basado en segmentos se probaron dos configuraciones diferentes que se explican en las siguientes subsecciones:

- Combinación ponderada de las cuentas de los modelos de traducción.
- Combinación lineal de los modelos de traducción.

Formalizando un poco más el procedimiento, si los datos de entrenamiento están formados por T conjuntos diferentes, podemos definir β_t el peso del conjunto t, para $1 \leq t \leq T$. El alineamiento a nivel de palabras lo estimaremos con Giza++ [Och & Ney, 2003] utilizando la concatenación de todo el material de entrenamiento disponible (en este caso Europarl, News-Commentary y United Nations). Después de esto, calcularemos la distribución de probabilidad de la traducción directa, inversa, y los modelos basados en IBM-1.

Combinación ponderada de las cuentas de los modelos de traducción

El peso β_t se aplica a la función $\mathcal{C}(\cdot)$, utilizada en las ecuaciones de los diccionarios léxicos (9.8) y (9.9), y las ecuaciones de las probabilidad de traducción de segmentos (9.13) y (9.14), que reescribimos aquí añadiendo el peso β_t:

$$q(y|x) = \frac{\sum_t \beta_t \mathcal{C}_t(x,y)}{\sum_t \beta_t \sum_{y'} \mathcal{C}_t(x,y')}, \quad (11.1)$$

$$q(x|y) = \frac{\sum_t \beta_t \mathcal{C}_t(x,y)}{\sum_t \beta_t \sum_{x'} \mathcal{C}_t(x',y)}, \quad (11.2)$$

$$p(y|x) = \frac{\sum_t \beta_t \mathcal{C}_t(\mathcal{L}(x,y))}{\sum_t \beta_t \sum_{\mathcal{L}(x,y')\in\Delta} \mathcal{C}_t(\mathcal{L}(x,y'))}, \quad (11.3)$$

$$p(x|y) = \frac{\sum_t \beta_t \mathcal{C}_t(\mathcal{L}(x,y))}{\sum_t \beta_t \sum_{\mathcal{L}(x',y)\in\Delta} \mathcal{C}_t(\mathcal{L}(x',y))}, \quad (11.4)$$

donde $\mathcal{C}_t(\cdot)$ es la función de conteo tomando como texto para extraer las cuentas el conjunto de entrenamiento t. Cuando una palabra o segmento no existe, su cuenta se fija a cero.

Combinación lineal de los modelos de traducción

En este caso, calculamos de forma independiente cada modelo de traducción para los T conjuntos de entrenamiento. El modelo de traducción final se obtiene interpolando linealmente cada uno de los modelos de traducción independientes. Los T modelos de traducción no tienen que compartir necesariamente los mismos pares de segmentos, y por tanto, cuando un par de segmentos no existe, se le asigna probabilidad cero.

El cómputo lo hacemos de la siguiente manera. Primero redefinimos las ecuaciones (9.8) y (9.9) para que calculen la distribución de probabilidad sobre los conjuntos de entrenamiento de forma separada. Esto obligará a tener $q_1, q_2, \ldots q_T$ diccionarios léxicos:

$$q_t(y|x) = \frac{\mathcal{C}_t(x,y)}{\sum_{y'} \mathcal{C}_t(x,y')}, \quad (11.5)$$

$$q_t(x|y) = \frac{\mathcal{C}_t(x,y)}{\sum_{x'} \mathcal{C}_t(x',y)}. \quad (11.6)$$

A partir de esto, podemos calcular los modelos léxicos tipo IBM-1 (ecuaciones (9.6) y (9.7)) como sigue:

$$\mathcal{H}'_{s2t}(x,y) = \sum_t \beta_t \frac{1}{(|x|+1)^{|y|}} \prod_{j=1}^{|y|} \sum_{i=0}^{|x|} q_t(y_j|x_i)\,, \tag{11.7}$$

$$\mathcal{H}'_{t2s}(x,y) = \sum_t \beta_t \frac{1}{(|y|+1)^{|x|}} \prod_{i=1}^{|x|} \sum_{j=0}^{|y|} q(x_i|y_j)\,, \tag{11.8}$$

y finalmente la probabilidad condicional de la traducción (ecuaciones (9.13) y (9.14)):

$$p(y|x) = \sum_t \beta_t \frac{\mathcal{C}_t(\mathcal{L}(x,y))}{\sum_{\mathcal{L}(x,y')\in\Delta} \mathcal{C}_t(\mathcal{L}(x,y'))}\,, \tag{11.9}$$

$$p(x|y) = \sum_t \beta_t \frac{\mathcal{C}_t(\mathcal{L}(x,y))}{\sum_{\mathcal{L}(x',y)\in\Delta} \mathcal{C}_t(\mathcal{L}(x',y))}\,. \tag{11.10}$$

Corpus

Se utilizó todo el corpus disponible de forma gratuita a través de la página web del WMT'11. La tabla 11.8 contiene las estadísticas del corpus bilingüe disponible para traducción automática. La parte correspondiente a la lengua destino (Español) se puede ver en la tabla 11.9. El corpus es preprocesado para homogeneizarlo, utilizando las herramientas puestas a nuestra disposición por la organización del WMT. Primero se trocea el texto y después se pasa todo a minúsculas. Esto implica que al final del proceso será necesario repetirlo pero a la inversa. Hay que volver pasar a mayúsculas, y detrocear. Este postproceso se implementó a través de la herramienta Moses y el "detokenizador" del WMT.

Entrenamiento de los modelos de lenguaje

El sistema baseline ha sido Moses, configurado con los parámetros estándar. El sistema incluye una combinación log-lineal de modelos formada por los dos modelos léxicos de traducción, los dos modelos de traducción condicional, un modelo de lenguaje de la lengua destino, la penalización debida al modelo de reordenamiento simple, el modelo lexicalizado de reordenamiento formado por seies componentes, y la respectiva penalización por número de palabras y número de segmentos producidos. Los pesos de esta combinación se optimizan con el procedimiento MERT (véase sección 1.7), utilizando como algoritmo de optimización el de Powell [Powell, 1964] que viene con las utilidades de la herramienta Moses. Como conjunto de desarrollo para ajustar los pesos del MERT se han utilizado los datos de News2008. El modelo de traducción baseline se ha estimado sobre la concatenación de los corpus bilingües de News-Commentary v6, United Nations y Europarl v6.

El modelo de lenguaje baseline es un modelo estándar de N-gramas estimado utilizando el suavizado Kneser-Ney. Concretamente, se ha entrenado un 6-grama para el corpus United Nations, un 5-grama para Europarl v6 y News-Shuffled, y un 4-grama para News-Commentary v6. Una vez estimados estos 4 modelos de lenguaje, se combinan de forma lineal minimizando la perplejidad en el conjunto de test de News2009. Para acelerar la decodificación, la talla del modelo final se ha reducido utilizando un umbral de poda de 10^{-8}.

Tabla 11.8: *Estadísticas del corpus bilingüe del WMT'11 calculadas tras limpiar el corpus, trocearlo y pasarlo a minúsculas. News2008 se usa como conjunto de desarrollo para el MERT, News2009 como conjunto de validación para la estimación de modelos de lenguaje, News2010 como test interno y News2011 como test final. //* English-Spanish WMT'11 corpus statistics, after cleaning, tokenization and lowercasing. News2008 is the development set (for MERT), News2009 is a tunning set (for language model optimization), News2010 is an internal test set, and News2011 is the official test set.

	English		Spanish		
Set	# Lines	# Words	# Lines	# Words	Voc. size
News-Commentary v6	107.0K	2.2M	107.0K	2.4M	60 336
Europarl v6	1.4M	28.8M	1.4M	29.8M	129 784
United Nations	7.5M	139.4M	7.5M	156.1M	409 957
Total	9.0M	170.4M	9.0M	188.4M	459 541
News2008	2.0K	49.7K	2.0K	52.6K	–
News2009	2.5K	65.6K	2.5K	68.0K	–
News2010	2.5K	61.9K	2.5K	65.5K	–
News2011	3.0K	74.8K	3.0K	79.4K	–

Tabla 11.9: *Estadística del corpus en Español para la tarea Inglés-Español del WMT'11. Todos los números han sido calculados tras trocear y pasar a minúsculas el texto. //* Spanish WMT'11 statistics after tokenization and lowercasing.

Set	# Lines	# Words
News-Commentary v6	159K	4.44M
News-Shuffled	9.17M	269M
Europarl v6	1.94M	55M
United Nations	6.22M	214M
Total	21.93M	678M

Tabla 11.10: *Pesos utilizados para las pruebas de combinación de modelos de traducción basada en segmentos. //* Weights used for the combination of phrase-based translation models.

System	Europarl v6	News-Commentary v6	United Nations
Smooth1	0.35	0.35	0.30
Smooth2	0.40	0.40	0.20
Smooth3	0.15	0.80	0.05
Linear	0.35	0.35	0.30

Se han probado tres combinaciones de pesos para la técnica de combinación de cuentas de los modelos de traducción. Para la técnica de interpolación lineal se han usado los pesos resultado de minimizar la perplejidad de los correspondientes modelos de lenguaje entrenados con Europarl v6, News-Commentary v6 y United Nations, usando el conjunto de test News2008. La tabla 11.10 resume los pesos utilizados en las pruebas.

Los NN LMs se entrenaron con todo el corpus descrito en la tabla 11.9, usando el Algorithm 2.1 on page 38 Los pesos elegidos para cada corpus son los mismos que los utilizados para combinar los modelos de lenguaje estándar. Para reducir la complejidad del modelo, el vocabulario de entrada del NN LM se ha restringido a aquellas palabras que aparecen más de 10 veces en el corpus. Esto nos da un vocabulario de entrada Ω^I formado por las 107 607 palabras más frecuentes, a las que se añaden dos entradas adicionales: una que representa las palabras fuera de este vocabulario, y otra para dar cuenta del context cue inicial (bcc). La salida del NN LM está formada por una Short-List con las 10K palabras más frecuentes, y dos salidas adicionales: una para aglutinar la probabilidad de las palabras fuera de la Short-List (OOS), y otra para dar cuenta del context cue final (ecc). Utilizamos un unigrama para distribuir la masa de probabilidad de la salida OOS sobre el conjunto de palabras fuera de la Short-List (véase la ecuación (3.5)).

El porcentaje de "running words" que quedan fuera de la Short-List del NN LM son un 10.7 % de las palabras del conjunto News2009 y un 11.1 % del conjunto News2011 (test oficial de la competición).

Se ha entrenado un 6-grama formado por la combinación lineal de 4 redes neuronales con diferentes valores de la proyección de cada palabra (128, 192, 256, 320). Cada red neuronal tiene 320 neuronas en la capa oculta. El reemplazamiento del conjunto de entrenamiento fue ajustado a 300K patrones en cada época de entrenamiento. El conjunto de validación ha sido News2009. El entrenamiento de las redes finalizó tras 99, 70, 53 y 42 épocas respectivamente. Esto implica que, en el mejor de los casos, la red neuronal ha entrenado con 29M patrones, frente a los 500M patrones que existen en el conjunto completo. La PPL alcanzada por la combinación de las 4 redes neuronales es de 281 en el conjunto News2009, frente a los 145 del N-grama estándar.

Resultados

Se han generado listas de N-best con 2 000 frases únicas. Los resultados de la repuntuación de estas listas se pueden ver en la tabla 11.11. Utilizando el test pairwise comparison, para un intervalo de confianza del 95 %, una diferencia de más de 0.3 puntos de BLEU es estadísticamente significativa. El sistema final obtuvo 31.5 puntos de BLEU, posicionándose en el segundo lugar de la clasificación (véase apéndice C, tabla C.1).

Tabla 11.11: *Resultados principales de la experimentación para la tarea Inglés-Español del WMT'11 con el test oficial. //* Main results for the English-Spanish WMT'11 task.

	News2010		News2011	
System	BLEU	TER	BLEU	TER
Baseline	29.2	60.0	30.5	58.9
Smooth1	29.3	59.9	–	–
Smooth2	29.2	59.9	–	–
Smooth3	29.5	59.6	30.9	58.5
+ NN LM	29.9	59.2	31.4	58.0
Linear	29.5	59.5	30.9	58.7
+ NN LM	**30.2**	**58.8**	**31.5**	**57.9**

Conclusiones

El sistema presentado logra una mejora de 0.4 puntos de BLEU tanto al combinar los modelos de traducción usando las cuentas (sistema Smooth3) como al combinarlos de forma lineal (sistema Linear). La incorporación de NN LMs en ambos sistemas añade una *mejora* de 0.5 puntos de BLEU para el sistema Smooth3, y de 0.6 puntos de BLEU para el sistema Linear. El resultado final del sistema presentado en el WMT'11 fue de 31.5 puntos de BLEU, lo que posiciona al sistema en *segundo lugar* en la clasificación.

La combinación de los modelos de traducción se puede mejorar optimizando los pesos β_t sobre el BLEU con un conjunto de desarrollo, en lugar de establecerlos manualmente. No obstante, ambas aproximaciones logran resultados similares en la experimentación aquí presentada.

11.3 Experimentos integrando NN LMs en la búsqueda

En este caso se participó en el "International Workshop of Spoken Language Translation" (IWSLT) en el año 2010 [Zamora-Martinez *et al.*, 2010], presentando, en la tarea BTEC Francés-Inglés, resultados comparativos entre Moses y el sistema de traducción presentado en el capítulo anterior. El IWSLT es una cita importante dentro del mundo de la traducción automática para intercambiar opiniones y tener una referencia comparada de la calidad de los sistemas desarrollados por los grupos de investigación. Se comparan resultados integrados con rescoring de listas de N-best. El apéndice C contiene la tabla C.2 con los resultados oficiales de esta evualación.

A modo comparativo también se han realizado experimentos utilizando el corpus News-Commentary 2010 del WMT'10, de mayor envergadura que la tarea BTEC Francés-Inglés del IWSLT.

11.3.1. Tarea BTEC Francés-Inglés del IWSLT'10

El IWSLT es otra de las citas importantes de la comunidad de traducción automática para establecer el posicionamiento de los sistemas de traducción desarrollados. A diferencia del

Tabla 11.12: *Estadísticas del corpus BTEC Francés-Inglés del IWSLT'10. Las estadísticas de los conjuntos de desarrollo se han calculado tras concatenar las* 16 *referencias múltiples. Train se refiere a la concatenación de BTEC+CSTAR'03, Dev2 al conjunto IWSLT'04 y Dev3 al conjunto IWSLT'05. //* Statistics of the French-English BTEC corpus of the IWSLT'10. The statistics of the development sets are computed after the concatenation of the 16 multiple references. Train refers to BTEC+CSTAR'03 sets, Dev2 refers to the IWSLT'04 set, and Dev3 refers to the IWSLT'05 set.

	French		English		
Set	# Lines	# Words	# Lines	# W	Voc. size
Train	28K	273K	28K	256K	7 599
Dev2	8K	72K	8K	67K	2 191
Dev3	8K	72K	8K	68K	2 271
Total	44K	417K	44K	392K	8 394
Test1	469	3 867	–	–	–
Test2	464	3 804	–	–	–

	N-gram bilingual translation corpus		
Set	# Lines	# Tuples	Voc. size
BTEC + CSTAR'03 (Train)	28K	214K	41 406

WMT, el IWSLT acentúa la importancia en la traducción del habla. No obstante, nuestra participación en la edición del 2010 se limitó a la traducción de texto.

La tabla 11.12 resume las estadísticas de la tarea BTEC Francés-Inglés del IWSLT'10, tras aplicar el proceso de troceado y de pasar a minúsculas todo el corpus, pero preservando los signos de puntuación. El troceado se ha realizado utilizando el script `tokenizer.perl` del WMT'10. El vocabulario del corpus Francés ha sido extraído de la partición de entrenamiento (9 954 palabras), y el vocabulario para el Inglés se ha extraído de la concatenación de todos los datos disponibles (8 394 palabras), a excepción de los conjuntos de test. La partición Dev2 se ha reservado para estimar y ajustar los parámetros de los modelos (combinación log-lineal, combinación de modelos de lenguaje). La partición Dev3 ha sido utilizada como test interno para seleccionar el mejor sistema. Finalmente, Test1 y Test2 son los conjuntos de test oficiales, ocultados por la organización del congreso.

Sistema baseline basado en N-gramas

El material de entrenamiento del baseline está formado por las 20K frases del conjunto de entrenamiento del BTEC así como las 16 referencias correspondientes al conjunto de desarrollo CSTAR'03. Se entrenó un modelo de traducción bilingüe de 3-gramas sobre la segmentación en tuplas bilingües del corpus. Para llevar a cabo la segmentación en tuplas se han alineado las palabras del corpus bilingüe utilizando la herramienta Giza++ [Och & Ney, 2003], configurada para utilizar el heurístico `grow-diag-final-and`. El modelo de reordenamiento lexicalizado se ha entrenado sobre el mismo conjunto de datos. Se ha entrenado un modelo de lenguaje de 3-gramas del inglés. Dicho modelo es una combinación lineal de dos modelos de lenguaje, uno del BTEC y otro del CSTAR'03. Ambos modelos han sido combinados minimizando la perplejidad en el conjunto Dev2.

Se ha utilizado una ventana de tamaño 6 para generar el grafo de reordenamiento siguiendo el heurístico local. El resto de parámetros se han dejado por defecto. El sistema baseline combina log-linealmente 7 o 13 modelos en función de si se usan o no los modelos de reordenamiento. Al extender el sistema con NN LMs se utilizan uno o dos modelos adicionales, dependiendo de si el NN LM se utiliza sólo para el modelo de lenguaje de la lengua destino, para el modelo de traducción bilingüe o para ambos. El sistema final, enviado a los organizadores para su evaluación (sistema primario), estuvo compuesto por 15 modelos.

Entrenamiento de los modelos de lenguaje

Todos los NN LMs se han entrenado siguiendo los detalles especificados en la parte I de esta tesis.

Para el modelo bilingüe de traducción ("Neural Network Translation Model" o NNTM) se ha entrenado un 3-grama NN LM a partir de una partición del conjunto de entrenamiento. Aleatoriamente se han utilizado 24K frases para entrenar y 4K frases para validar el entrenamiento y establecer el criterio de parada. El NNTM es una combinación lineal de 4 redes neuronales con distinto número de unidades por palabra en la capa de proyección $(128, 160, 192, 256)$, y con 200 unidades en la capa oculta. Como Short-List se ha utilizado el vocabulario formado por las 9K tuplas con mayor frecuencia de aparición (en total hay $|\Delta| = 41$K tuplas). Dicha Short-List ha sido utilizada tanto a la entrada como a la salida de las redes neuronales.

Un 4-grama NN LM se ha entrenado como modelo de lenguaje de la lengua destino ("Neural Network Target Language Model" o NNTLM). En este caso se ha utilizado todo el conjunto de entrenamiento para entrenar la red y el conjunto Dev2 para validación y criterio de parada. Del mismo modo el NNTLM es una combinación lineal de 4 redes neuronales con las mismas características que antes, $128, 160, 192, 256$ unidades de proyección para cada palabra, y 200 unidades en la capa oculta. La Short-List estuvo formada por las 5K palabras más frecuentes de un vocabulario completo de $|\Omega| = 8$K palabras.

Los modelos de lenguaje están integrados de forma acoplada en el proceso de decodificación utilizando la aproximación Fast NN LM descrita en el capítulo 4. Para cada posible valor de N el modelo de lenguaje está formado por los modelos conexionistas de lenguaje que van desde el mayor orden hasta el bigrama. El precómputo de las constantes de normalización softmax permite acelerar la evaluación de los modelos. Cuando no se encuentra una constante, se utiliza el siguiente modelo de menor orden, y se repite este proceso hasta llegar al bigrama conexionista.

Resultados

En la tabla 11.13 se encuentran los resultados de traducir el conjunto Dev2 con todas las configuraciones probadas. Todos estos resultados están en el formato `case+punc` especificado por la organización del IWSLT. Se ha utilizado Moses para llevar a cabo el proceso de *recasing*. En los resultados se han evaluado tres factores:

- **Efecto del modelo de reordenamiento lexicalizado.** Para el sistema baseline, usar el modelo de reordenamiento lexicalizado sólo aporta una pequeña mejora de 0.2 puntos de BLEU, similar a la mejora obtenida al utilizar NN LMs en la etapa de rescoring de N-best. No obstante, cuando se integran los NN LMs en el proceso de decodificación, se logra una *mejora* de 0.8 puntos de BLEU.

Tabla 11.13: *Resultados en BLEU para los conjuntos Dev2 (desarrollo) y Dev3 (test interno) de la tarea BTEC Francés-Inglés del IWSLT'10. Se ha medido el número de palabras por segundo procesadas por cada sistema. //* BLEU for the Dev2 and Dev3 sets from the French-English BTEC IWSLT'10 task.

System	Dev2	Dev3	Wrds/Sec.
Moses	64.3	64.3	–
+ NNTLM	65.4	65.7	–
N-gram-based	65.8	65.2	21
+ R	65.8	65.4	21
Integrating NN LMs in the decoder			
+ NNTLM	67.0	65.7	8
+ NNTM	66.6	65.4	10
+ NNTLM + NNTM	67.4	66.3	7
+ NNTLM + NNTM + R	67.7	67.1	7
Rescoring 1000-best list			
+ NNTLM + NNTM	66.9	66.4	–
+ NNTLM + NNTM + R	67.8	66.6	–

- **Efecto del uso de NNTLM y NNTM.** Añadir el NNTLM a la combinación log-lineal logra una mejora de 0.5 puntos de BLEU sobre el baseline. La adición de NNTM alcanza una mejora menor, de 0.2 puntos. No obstante, añadir ambos modelos, NNTLM y NNTM, alcanza una mejora de 1.1 puntos de BLEU. Si además añadimos el modelo de reordenamiento, la *mejora* llega hasta los 1.7 puntos de BLEU.

- **Efecto de NN LMs integrados durante la decodificación frente al rescoring de listas de N-best.** Los resultados son muy similares en ambos escenarios si no se utiliza el modelo de reordenamiento lexicalizado. No obstante, al añadir el modelo de reordenamiento se amplían las diferencias, llegando a ser de 0.5 puntos de BLEU. Esto se puede explicar por una mejor optimización del procedimiento Mert. Para esta tarea hemos utilizado listas de N-best con repeticiones, para permitir que el modelo de traducción pueda seleccionar la segmentación en tuplas que sea mejor.

  Tras analizar los resultados obtenidos en otras tareas durante esta tesis, podemos afirmar que realmente no hay diferencias significativas entre integrar los modelos o hacer rescoring de listas de N-best, siempre y cuando las listas sean de al menos $1\,000$ hipótesis únicas, sin repetición, aunque en el caso de sistemas de traducción basados en N-gramas se aprecian resultados mejores con sistemas integrados que con rescoring de listas de N-best.

Resultados oficiales

La tabla 11.14 muestra los resultados oficiales en BLEU para los conjuntos de test ofrecidos por la organización (Test1 y Test2 en la tabla 11.12). El primer conjunto de resultados utiliza los mismos sistemas que los mostrados en la tabla 11.13. Para el segundo conjunto de resultados se han modificado los modelos añadiendo todos los datos disponibles (BTEC training + 4 conjuntos de desarrollo con 16 referencias múltiples). Los pesos de la combinación

Tabla 11.14: *Resultados oficiales en BLEU para los conjuntos de test IWSLT'09 (Test1) y del IWSLT'10 (Test2). El sistema primario se ha marcado en negrita. Todos los resultados son con el sistema integrado. //* BLEU for the official test sets from the IWSLT'09 (Test1) and IWSLT'10 (Test2). All results are with the integrated system.

System	Test1	Test2
Moses	59.8	51.2
N-gram-based + R	61.4	52.2
+ NNTLM + NNTM + R	62.5	54.3
Adding all available data		
Moses	59.7	51.2
+ NNTLM	60.9	53.6
N-gram-based + R	61.8	51.2
+ NNTLM + NNTM + R	**63.6**	**53.6**

log-lineal son iguales para ambos conjuntos. Los modelos de lenguaje del Inglés han sido reentrenados de la siguiente forma:

- En el caso de los modelos estándar de N-gramas se ha estimado un modelo de lenguaje para cada conjunto de datos disponible. El coeficiente de la combinación lineal de estos modelos ha sido calculado mediante validación cruzada particionando cada conjunto en 10 bloques.
- Para los NN LMs (NNTM y NNTLM) se ha utilizado la versión estocástica del algoritmo de entrenamiento. Esta configuración permite establecer un peso para cada conjunto de entrenamiento. Estos pesos son idénticos a los utilizados para combinar los modelos de lenguaje estándar.

Podemos observar en los resultados finales que la adición de NN LMs a la combinación log-lineal logra una mejora muy importante. Por otro lado, no está clara la ventaja de añadir todos los datos disponibles, si bien es posible que los resultados no sean todo lo buenos que podrían ser debido a los coeficientes de la combinación log-lineal o al método utilizado para combinar los modelos de lenguaje. El resultado oficial para el test09 es de 63.6 puntos de BLEU, y para el test10 es de 53.6 puntos de BLEU, lo que supone una mejora de 1.8 y 2.4 puntos de BLEU sobre el sistema baseline. Estos resultados son estadísticamente significativos aplicando el test de comparación entre sistemas (pairwise comparison) de la sección 1.6.

A modo comparativo se han estimado modelos de traducción con la herramienta Moses, cuyos resultados se muestran en las tablas 11.13 y 11.14.

Si comparamos la velocidad de los diferentes sistemas podemos observar que la adición de los NN LMs incrementa entre dos y tres veces el coste de la traducción (tabla 11.13).

Conclusiones

A partir de estos resultados podemos concluir que el uso de NN LMs, tanto para el modelo de lenguaje de la lengua destino como para el modelo bilingüe de lenguaje, permite *mejorar de forma sustancial* los resultados del sistema.

A nivel de calidad del resultado, la aproximación integrada y la aproximación basada en el rescoring de listas de N-best muestran *diferencias no significativas* en los resultados, observándose en esta tarea una diferencia de 0.5 puntos de BLEU. No obstante, tener un sistema totalmente integrado es una ventaja en sí mismo, pues permite que la arquitectura del sistema sea más sencilla. El sistema presentado quedó en *segundo lugar* utilizando el criterio de evaluación de la organización del IWSLT'10 (véase apéndice C, tabla C.2).

Desde la perspectiva del coste computacional hemos observado que integrar los NN LMs incrementa entre *dos y tres veces* el coste. Por tanto, el algoritmo de traducción desarrollado necesita mejorar su eficiencia computacional. Como ya se ha comentado en la sección 10.8, el algoritmo de Viterbi para grafos utilizado es una adaptación del mismo algoritmo utilizado para reconocimiento de secuencias con HMMs. Sin embargo, en traducción automática los grafos son *multietapa*, una característica que ha quedado por explotar en el algoritmo, si bien la arquitectura ha sido preparada para llevar a cabo esta modificación, que queda pendiente como trabajo futuro.

11.3.2. Tarea News-Commentary 2010 Español-Inglés

El objetivo de esta sección es estudiar el efecto de los NN LMs integrados en la decodificación con una tarea de mayor envergadura. Se estudia tanto el efecto del modelo de lenguaje de la lengua destino como el del modelo bilingüe de traducción. Se comparan resultados con diferentes NN LMs integrados en la búsqueda y haciendo rescoring de listas de las N-best, y se contrasta la aproximación basada en segmentos y la basada en N-gramas. El tamaño y dispersión del corpus bilingüe, utilizado para entrenar el modelo de traducción de N-gramas, precisó la estimación de NN LMs de clases estadísticas. Los datos del corpus utilizado se muestran en las tablas 11.15 y 11.16, tras aplicar el troceado (usando la herramienta `tokenizer.perl` del WMT'10) y pasar a minúsculas el corpus. Los resultados de las medidas de evaluación (BLEU y TER) mostrados en las tablas de esta sección son con los resultados troceados y en minúsculas.

Sistema baseline basado en N-gramas

El material de entrenamiento del baseline está formado por las 80.9K frases del conjunto News-Commentary 2010. Se entrenó un modelo de traducción bilingüe de 4-gramas sobre la transformación y segmentación en tuplas bilingües del conjunto. El procedimiento de extracción y segmentación en tuplas es el descrito en la sección 9.4.1. Para dicho cometido se utiliza el alineamiento producido por la herramienta Giza++ [Och & Ney, 2003] configurada para utilizar el heurístico `grow-diag-final-and`. El modelo de reordenamiento lexicalizado se ha entrenado sobre el mismo conjunto de datos. Se ha entrenado un modelo de lenguaje de 4-gramas del inglés como baseline. El sistema ha sido optimizado a través del procedimiento MERT (véase sección 1.7) utilizando el conjunto News2008 para la optimización.

Se ha usado una ventana de tamaño 5 para generar el grafo de reordenamiento siguiendo el heurístico local. El resto de parámetros se han dejado por defecto. Todos los sistemas incorporan cálculo del coste futuro para mejorar la poda del grafo de reordenamiento y los seis modelos de reordenamiento lexicalizados.

Tabla 11.15: *Estadísticas del corpus bilingüe News-Commentary 2010, tras limpiar, trocear y pasar a minúsculas. News2008 se usa como conjunto de desarrollo, News2009 como test interno y News2010 como test final. //* Statistics of the bilingual corpus News-Commentary 2010 extracted from the WMT'10, after cleaning, tokenization and lowercasing. News2008 is the development set, News2009 is an internal test set and News2010 is the final test set.

	Spanish		English		
Set	# Lines	# Words	# Lines	# Words	Voc. size
News-Commentary 2010	80.9K	1.8M	81.0K	1.6M	38 781
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	–

	N-gram bilingual translation corpus		
Set	# Lines	# Tuples	Voc. size
News-Commentray 2010	80.9K	1.5M	231 981

Tabla 11.16: *Estadísticas de la parte inglesa de la tarea News-Commentary 2010. Todos los números han sido calculados tras trocear y pasar a minúsculas el texto. //* English statistics of the News-Commentary 2010 corpus after tokenization and lowercasing.

Set	# Lines	# Words
News-Commentary 2010	125.9K	2.97M

Entrenamiento de los modelos de lenguaje

Para el modelo de lenguaje de la lengua destino (NNTLM) se han probado NN LMs de orden $N = 2, 3, 4, 5$, que identificaremos con el nombre NNTLM-Ngr, donde N es el orden. Cada NNTLM es una combinación lineal de tres redes neuronales que difieren en el tamaño de la capa de proyección para cada palabra (128, 160 y 208 neuronas), y con 200 neuronas de capa oculta. Se ha utilizado una Short-List de las 20K palabras más frecuentes, que sirve tanto para el vocabulario de entrada Ω^I como para el vocabulario de la salida Ω'. La masa de probabilidad de las palabras que faltan hasta completar el vocabulario Ω de la tarea, cuya talla es 39K, se calcula a través de la neurona OOS utilizando la aproximación descrita en la sección 3.1.2.

El modelo bilingüe de traducción (NNTM) es más problemático en esta tarea que en los experimentos de la sección 11.3.1 debido al tamaño del vocabulario de tuplas $|\Delta| \approx 232$K. Dicho tamaño implica que, si se utiliza una Short-List de las 20K tuplas más frecuentes, el número de tuplas fuera de dicha Short-List en el conjunto de entrenamiento asciende al $84\,\%$ de las 1.5M de tuplas existentes. Esta proporción imposibilita la estimación de un modelo conexionista de lenguaje utilizando estas ideas. Para solucionar este problema sería posible probar la eficacia del modelo conocido como "Structured Output Layer NN LM" [Hai-Son *et al.*, 2011], ya que permite estructurar la salida de la red neuronal para poder calcular la probabilidad de todo el vocabulario sin utilizar una Short-List. No obstante, probar este modelo queda pendiente como trabajo futuro. La solución adoptada es más simple y se basa en la estimación y clasificación de las tuplas en *clases estadísticas*.

Entrenamiento del NNTM de clases estadísticas

El NNTM de clases estadísticas se estima siguiendo este proceso:

1. A través del comando `mkcls` de la herramienta Giza++ [Och & Ney, 2003] se distribuyen las tuplas del conjunto de entrenamiento en CLS clases estadísticas [Och, 1999], siendo dicha distribución en clases no ambigua (una tupla no puede estar en más de una clase). CLS es un nuevo parámetro a estimar de forma empírica.

2. Se calcula la probabilidad de pertenencia de las tuplas a las clases de esta manera:

$$p(z \in \Delta | c \in CLS) = \frac{\mathcal{C}(z|c)}{\sum_{z' \in \Delta} \mathcal{C}(z'|c)}\,, \tag{11.11}$$

   siendo $\mathcal{C}(z|c)$ el número de veces que aparece la tupla z en la clase c.

3. Se sustituye cada tupla del conjunto de entrenamiento por su clase correspondiente, dando lugar a un nuevo conjunto de entrenamiento.

4. Se estima un modelo conexionista de lenguaje sobre dicho conjunto de entrenamiento.

5. En tiempo de evaluación, la probabilidad condicional del modelo bilingüe de traducción se aproxima siguiendo esta ecuación:

$$p(\mathcal{T}_i | \mathcal{T}_{i-1} \ldots \mathcal{T}_{i-N+1}) \approx p(\mathcal{T}_i | c_i) \cdot p(c_i | c_{i-1} \ldots c_{i-N+1})\,, \tag{11.12}$$

   siendo c_i la clase a la que pertenece la tupla $\mathcal{T}_i$. Por tanto, se estima la probabilidad condicional aproximándola al producto entre:

- la probabilidad condicional de la tupla actual dentro de su clase $p(\mathcal{T}_i|c_i)$;
- la probabilidad condicional de la clase de la tupla actual dada la secuencia de clases de las tuplas anteriores $p(c_i|c_{i-1}\dots c_{i-N+1})$.

Esta aproximación permite estimar el modelo conexionista de lenguaje incluso aun cuando el vocabulario de la tarea es tremendamente disperso, como sucede con los modelos bilingües de traducción para tareas con vocabularios grandes.

Siguiendo el procedimiento aquí descrito, se han entrenado NNTMs utilizando distintos valores del orden $N = 2, 3, 4, 5$ y del número de clases $CLS = 100, 300, 500, 1000$. No se han probado todas las combinaciones, sólo aquellas más prometedoras. De nuevo, cada NNTM es la combinación de tres redes neuronales, utilizando valores de capa de proyección por palabra de $64, 96, 128$ y 200 neuronas en la capa oculta. El vocabulario de la entrada Ω^I y de la salida de la red neuronal Ω' es el mismo, y se corresponde con el vocabulario de clases elegido en cada caso.

El sistema de traducción integra los modelos de lenguaje de forma totalmente acoplada en el proceso de decodificación, por lo que se han utilizado los modelos smoothed Fast NN LM descritos en el capítulo 4. Para cada valor de N el modelo de lenguaje está formado por los modelos conexionistas de lenguaje que van desde el mayor orden hasta el bigrama, utilizando la idea de precomputar las constantes de normalización softmax para acelerar la evaluación de los modelos. Cuando no se encuentra una constante, se utiliza el siguiente modelo de menor orden, y así sucesivamente hasta llegar al bigrama conexionista.

Resultados

La tabla 11.17 muestra los resultados del sistema baseline basados en N-gramas (`April-NB`) utilizando NN LMs para la lengua destino con distintos valores de N. También se pueden observar resultados de PPL de cada uno de los modelos de lenguaje. Nótese que la PPL se ha calculado combinando linealmente los NN LMs con el modelo estadístico, no obstante el sistema de traducción los combina a través de la combinación log-lineal. El sistema de traducción es integrado, por tanto utiliza el modelo smoothed Fast NN LM, sin embargo, a efectos comparativos se ha calculado también la PPL obtenida por el modelo on-the-fly Fast NN LM.

Los resultados se han obtenido integrando los modelos durante el proceso de decodificación, y el mejor resultado es alcanzado por el modelo con $N = 4$, logrando 20.9 puntos de BLEU y $59.9\,\%$ de TER. La mejora respecto al baseline es de 0.7 puntos absolutos de BLEU ($3.5\,\%$ de mejora relativa) y de 0.5 puntos absolutos de TER ($0.8\,\%$ de mejora relativa). La mejora obtenida en PPL es de 43 puntos absolutos ($16\,\%$ de mejora relativa).

La figura 11.2 y la tabla 11.18 ilustran los distintos NNTMs probados. Se observa que el mejor modelo es el NNTM-300-4gr, cuyos parámetros son $CLS = 300$ y $N = 4$. La mejora obtenida sobre el baseline es de 0.7 puntos absolutos de BLEU ($3.5\,\%$ de mejora relativa, igual que con el mejor de los NNTLMs) y de 0.4 puntos absolutos de TER ($0.6\,\%$ de mejora relativa, peor que el mejor de los NNTLMs).

Finalmente, la tabla 11.19 ilustra el resultado de combinar el baseline con el mejor de los NNTLMs y los dos mejores NNTMs. Se observa que el efecto de la combinación es mejor para el modelo NNTM-500-4gr, que es el segundo mejor de los NNTMs probados con el conjunto de desarrollo News2009. La mejora que se obtiene respecto al baseline es de 1 punto absoluto de BLEU ($5\,\%$ de mejora relativa) y de 0.7 puntos absolutos de TER ($1.2\,\%$ de

Tabla 11.17: *Resultados en BLEU/TER y PPL para el conjunto de desarrollo News2009 y distintos valores de N para modelos de lenguaje de la lengua destino. //* BLEU/TER and PPL for the News2009 development set and different N values for the NNTLM. PPL refers to an hypothetic linear combination of the NN LM and the standard N-gram, however, the translation system combines them in a log-linear way.

System	News2009 BLEU	TER	PPL	
`April`-NB baseline	20.2	60.4	269	
			Fast NN LM	on-the-fly Fast NN LM
+ NNTLM-2gr	20.3	60.4	246	246
+ NNTLM-3gr	20.6	60.2	231	227
+ NNTLM-4gr	20.9	59.9	226	217
+ NNTLM-5gr	20.8	60.0	225	213

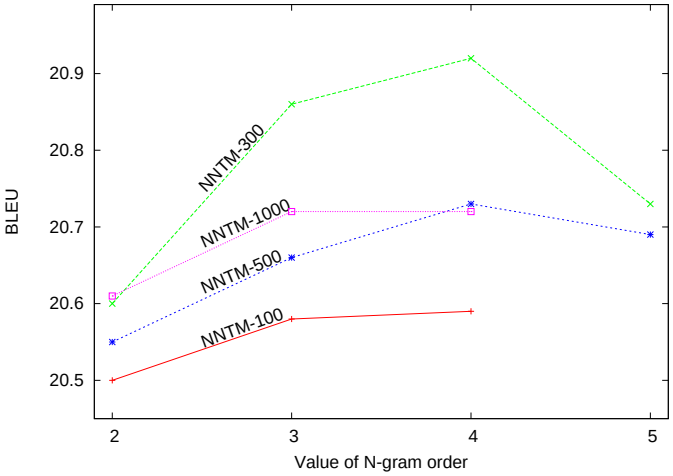


Figura 11.2: *Resultados en BLEU para el conjunto de desarrollo News2009 y diferentes valores de N y CLS de los modelos NNTM usando el sistema `April`-NB. //* BLEU plot for the News2009 development set and different values of N and CLS of the NNTMs.

Tabla 11.18: *Mejores resultados en BLEU/TER para el conjunto de desarrollo News2009 y distintos valores de N y CLS para modelos bilingües de traducción. //* Best BLEU/TER for the News2009 development set and different values of N and CLS.

System	News2009 BLEU	TER
`April`-NB baseline	20.2	60.4
+ NNTM-100-4gr	20.6	60.0
+ NNTM-300-4gr	20.9	60.0
+ NNTM-500-4gr	20.7	60.0
+ NNTM-1000-3gr	20.7	60.1

Tabla 11.19: *Resultados en BLEU/TER para el conjunto de desarrollo News2009 combinando los mejores NNTM con el modelo NNTLM-4gr. //* BLEU/TER for the News2009 development set combining the two best NNTMs with the NNTLM-4gr.

System	News2009 BLEU	TER
April-NB baseline	20.2	60.4
+ NNTLM-4gr	20.9	59.9
+ NNTM-300-4gr	21.1	59.7
+ NNTM-500-4gr	21.2	59.7

mejora relativa). Los resultados en BLEU son significativos bajo el test pairwise comparison para un intervalo de confianza del $95\,\%$.

Resultados finales

Finalmente, en la tabla 11.20 se encuentran los resultados con el conjunto de validación News2009 y el conjunto de test News2010. Se han probado distintas configuraciones baseline para ampliar el análisis de resultados y la comparación entre Moses y el sistema de traducción desarrollado en esta tesis:

- Moses baseline: configuración por defecto de Moses utilizando exactamente los mismos datos de entrenamiento y conjuntos de evaluación.
- April-PB: el sistema de traducción del capítulo anterior, configurado para ser utilizado con modelos basados en segmentos, y usando los modelos de traducción de Moses. El procedimiento del MERT es ejecutado utilizando April en lugar de Moses.
- Moses*: la herramienta Moses configurada usando los pesos de la combinación log-lineal calculados con el MERT de April-PB.
- April-NB: el sistema de traducción del capítulo anterior, configurado para trabajar con modelos de traducción basados en N-gramas.

Comparando los sistemas baseline. Podemos observar que las diferencias entre Moses y la herramienta April no son significativas en ninguno de los casos probados (Moses, April-PB, Moses*, April-NB). Esta comparativa permite reafirmar la validez del algoritmo de búsqueda implementado en April, y descrito en el capítulo anterior, comparándolo con resultados de traducción reconocidos como estado del arte.

Usando NN LMs en el sistema totalmente integrado en la decodificación. La adición de los modelos conexionistas de lenguaje de traducción y de lengua destino (NNTM y NNTLM) en el sistema April-NB alcanza una *mejora* de 0.9 puntos absolutos de BLEU ($4\,\%$ de mejora relativa) y de 0.9 puntos absolutos de TER ($1.6\,\%$ de mejora relativa) en el conjunto de test News2010. Estas diferencias son significativas bajo el test pairwise comparison usando una confianza del $95\,\%$. Si lo comparamos con el sistema April-PB al añadirle el NNTLM, observamos que el sistema basado en N-gramas mejora al basado en segmentos en 0.4 puntos

Tabla 11.20: *Resultados finales en BLEU/TER para el conjunto de desarrollo News2009 y el test News2010 de la tarea News-Commentary 2010. NNTM es NNTM-500-4gr y NNTLM es NNTLM-4gr. //* Final BLEU/TER for the News2009 development and the News2010 test set, with NNTM-500-4gr and NNTLM-4gr.

	News2009 (dev.)		News2010 (test)	
System	BLEU	TER		TER
Moses baseline	20.4	60.3	22.6	57.8
April-PB	20.6	60.3	22.7	57.8
Moses*	–	–	22.6	57.9
April-NB	20.2	60.4	22.7	58.0
Integrating NN LMs in the decoder				
April-PB + NNTLM	21.2	59.8	23.2	57.5
April-NB + NNTLM	20.9	59.9	–	–
April-NB + NNTM	20.7	60.0	–	–
April-NB + NNTLM + NNTM	**21.2**	**59.7**	**23.6**	**57.1**
Rescoring 2000-uniq-best list				
April-PB + NNTLM	21.1	59.9	23.4	57.3
April-NB + NNTLM	20.9	60.0	–	–
April-NB + NNTM	20.6	60.2	–	–
April-NB + NNTLM + NNTM	21.1	59.8	23.5	57.4

absolutos de BLEU y 0.4 puntos absolutos de TER. Sin embargo, con el conjunto de validación News2009 los resultados son idénticos para ambos sistemas, lo que significa que el sistema basado en N-gramas logra una *generalización* relativamente mayor. Por otro lado, si comparamos el sistema April-NB usando sólo el NNTLM o bien sólo el NNTM con el sistema April-PB, los resultados con el conjunto de validación News2009 muestran un empeoramiento de los resultados respecto al sistema basado en segmentos.

Usando NN LMs en una etapa de rescoring con listas de 2000-uniq-best. De nuevo, los *mejores* resultados haciendo rescoring de N-best son obtenidos por el sistema April-NB que incorpora tanto NNTMs como NNTLMs. Hay que destacar que en este caso se utiliza la aproximación on-the-fly Fast NN LM en lugar de la smoothed Fast NN LM, con lo que los resultados son los mismos que obtendría un NN LM estándar. El sistema basado en N-gramas empeora sus resultados respecto a la versión totalmente integrada, tanto con el conjunto de validación News2009 como con el conjunto de test News2010. El sistema April-PB, que usa modelos de traducción basada en segmentos, no adolece de este problema y se observa una mejora de 0.2 puntos absolutos de BLEU y de 0.2 puntos absolutos de TER respecto al sistema integrado, como cabría esperar a priori ya que los modelos smoothed Fast NN LM son una versión degrada del on-the-fly Fast NN LM. Esta diferencia entre el sistema basado en N-gramas y el basado en segmentos precisa de una mayor investigación, si bien la hipótesis más plausible es que el espacio de búsqueda de los sistemas basados en segmentos está mejor representado en una lista de N-best sin repetición, mientras que el espacio de búsqueda de un sistema basado en N-gramas precisa de listas de N-best más grandes.

Conclusiones

Se han entrenado modelos conexionistas de lenguaje para una tarea de tamaño medio logrando alcanzar mejoras sistemáticas en BLEU y TER. Se han probado diferentes combinaciones de orden N y de modelos de lenguaje de clases para disminuir el efecto del problema de la dispersión de los datos en el modelo bilingüe de traducción. Se ha logrado una mejora de 0.9 puntos absolutos de BLEU (mejora relativa del $4\,\%$) sobre el mejor de los baselines. Esta diferencia es significativa bajo el test pairwise comparison para un intervalo de confianza del $95\,\%$.

A efectos comparativos se han estudiado tanto sistemas basados en segmentos (`April`-PB) como sistemas basados en N-gramas (`April`-NB), observando que el sistema `April`-NB logra una mayor generalización gracias al uso de dos modelos conexionistas de lenguaje, el modelo conexionista de traducción (NNTM) y el modelo conxionista de la lengua destino (NNTLM). La integración de los modelos conexionistas de lenguaje en el proceso de decodificación, frente a realizar un rescoring de listas de N-best, ha resultado satisfactoria para el caso de `April`-NB, mientras que ha empeorado ligeramente los resultados para el sistema `April`-PB.

Para obtener estas mejoras ha sido necesario entrenar el NN LM usado para el modelo bilingüe de traducción sobre el corpus etiquetado con clases estadísticas, ya que el tamaño del vocabulario de tuplas y la dispersión del mismo impiden utilizar la aproximación de la Short-List. Esta aproximación ha sido satisfactoria ya que ha logrado mejorar el baseline. Sin embargo, se ha dejado como trabajo futuro utilizar una red tipo Structured Output Layer (SOUL) NN LM [Hai-Son *et al.*, 2011] que promete resolver el problema de la talla del vocabulario a través de una estructura jerárquica de la salida de la red neuronal.

11.4 Resumen

Este capítulo resume los resultados experimentales de esta tesis utilizando NN LMs en sistemas de traducción automática estadística. Los experimentos se han organizado en dos grandes grupos:

- Sistemas desacoplados basados en herramientas de traducción estándar, añadiendo una segunda etapa de rescoring de listas de N-best con los NN LMs desarrollados en esta tesis.
- Sistemas donde los NN LMs se integran en el proceso de búsqueda, utilizando el algoritmo de traducción descrito en el capítulo anterior.

Hemos encontrado que en todos los casos, el uso de NN LMs permite mejorar la calidad del sistema utilizado, independientemente de si estos son integrados o no en la búsqueda, con mejoras que van desde 0.5 puntos de BLEU hasta 2.4 puntos de BLEU en función de la tarea. En todo caso las diferencias en BLEU han resultado ser significativas para el test pairwise comparison con una confianza del $95\,\%$.

El sistema integrado ha logrado mejorar ligeramente los resultados del sistema basado en N-gramas, tanto para tareas pequeñas como la BTEC del IWSLT'10, y para tareas de tamaño medio como News-Commentary 2010. No obstante, a nivel computacional todavía queda trabajo que hacer, pues el sistema es entre 2 y 3 veces más lento con NN LMs. Sin embargo, en esta tesis se ha observado que en tareas de reconocimiento de escritura manuscrita (véanse

los experimentos de la sección 4.5) ha sido posible lograr que la eficiencia de la integración sea mucho mayor. Este hecho motiva la necesidad de mejorar el sistema de traducción en la línea explicada en la sección 10.8, explotando la característica multietapa de los grafos de traducción automática aprovechando la señal `change_stage` del protocolo de incidencia para serialización de grafos.

Como objetivos científicos, este capítulo ha presentado:

- Experimentación que valida los buenos resultados de los NN LMs en traducción automática a través del rescoring de listas de N-best. Estos resultados se encuentran publicados en varios congresos y workshops [Khalilov *et al.*, 2008b], [Khalilov *et al.*, 2008a], [Zamora-Martínez & Sanchis, 2010], [Zamora-Martinez & Castro-Bleda, 2011].
- Se ha presentado un método para mejorar la combinación de modelos de traducción basada en segmentos, publicado en [Zamora-Martinez & Castro-Bleda, 2011].
- Evaluación del sistema de traducción basada en N-gramas que integra NN LMs en el proceso de búsqueda, obteniendo resultados muy competitivos. Este sistema fue publicado en [Zamora-Martinez *et al.*, 2010].
- Ampliación de los resultados del punto anterior a una tarea de mayor envergadura. Se han estimado modelos bilingües de traducción tras una clasificación de las tuplas en clases estadísticas no ambiguas, mostrando que es posible mejorar los resultados del sistema aplicando esta idea. Estos resultados, junto con los de la tarea anterior, han sido enviados al congreso COLING 2012 [Zamora-Martinez & Castro-Bleda, 2012].

Parte IV

Conclusions

12

CONCLUSIONS AND FUTURE WORK

Índice

This chapter presents the conclusions and contributions of this thesis, and synthesizes the future work exposed before. The publications related to the work will be found at the end of this chapter, chronologically and by relevance ordered.

12.1 Conclusions and contributions

The presentation of this work has attempted to guide the reader through the most relevant contributions of it to the connectionist language modeling field and its application to:

- spoken language understanding;
- handwritten text recognition;
- statistical machine translation.

The common thread in the three fields is connectionist language modeling. Very significant improvements were achieved in every task using this kind of language model.

This thesis formalizes the Neural Network Language Models (NN LMs), highlighting the importance of the combination of NN LMs and their standard N-gram counterpart. Nowadays, there exists an important increasing interest in the scientist community to enhance this combination.

12.1.1. Contributions to connectionist language modeling

The most relevant contributions to the field of connectionist language modeling are the formalization and empirical evaluation of the speed-up technique that has allowed the integration of NN LMs into the decoding process. To summarize, the main contributions are:

1. Formalization and development of a *totally coupled* Viterbi decoding process working on lattices and NN LMs. This formalization is built over a trade-off solution for the Short-List problem, and over a description of NN LMs as full stochastic finite state machines.

2. Formalization of the speed-up technique based on the precomputation of the softmax normalization constants. Experimental evaluation of this technique was published in [Zamora-Martínez *et al.*, 2009] and referenced by the ScholarPedia [Bengio, 2008].

3. An extension of the NN LMs with an input *cache* module in order to adapt the language model probability distribution. This cache stores a summary of past human-machine interactions. Different approaches are proposed to apply this idea to different tasks:

   - For spoken language understanding, in a dialog task, the cache stores a summary of the dialog state. Experiments on the French Media task, using these ideas, were published in [Zamora-Martínez *et al.*, 2012a].
   - In handwritten text recognition tasks, a cache can summarize the last document chapter, or the last k pages, of the current transcribed page. This experimentation is proposed as a future work.
   - In statistical machine translation tasks, the cache can be composed of the previous translated sentences of the current document, and in addition, the previous source language sentences of the current document. This is also proposed as future work.

   Three information sources are differentiated:

- Known data: for instance, the source language sentence in machine translation systems.
- Probably erroneous data: for instance, the previous best hypothesis decoded by the system.
- Data with dependencies: for instance, the machine sentences generated by a dialog system to answer user interactions.

12.1.2. Contributions to sequence recognition

Two major contributions were presented in the sequence recognition field. First, an empirical evaluation of NN LMs with cache (Cache NN LMs). Second, an exhaustive experimentation with NN LMs in handwritten text recognition:

1. Empirical evaluation of Cache NN LMs in a spoken language understanding system has showed that Cache NN LMs achieve the performance of complex state-of-the-art SLU systems. The system achieves a concept error rate of $26.1\,\%$ points (a $10\,\%$ of relative improvement over the proposed baseline without cache memory). These results were published in [Zamora-Martínez *et al.*, 2012a].

2. NN LMs were integrated into the HMM decoding process using a Viterbi two-level algorithm. Important improvements were achieved in word error rate of a standard handwritten text recognition task (1.7 absolute points of WER, $7.7\,\%$ relative improvement over the baseline). Computational cost of NN LMs was reduced drastically, being similar to the computational cost of a standard system. Some of these results are expected to be published in the "Pattern Recognition" journal, where a contribution was submitted [Zamora-Martínez *et al.*, 2012c].

3. Character-based language models, instead of word-based language models, were empirically evaluated on the IAM-DB handwritten text recognition task. NN LMs achieve an improvement of 6 absolute WER points over an standard language model ($21.6\,\%$ of relative improvement), and 4 absolute character error rate points ($26.8\,\%$ of relative improvement). Character-based language models recover the $33\,\%$ of words that are unknown for word-based models. This research line is very interesting in order to transcribe old documents for which language resources are limited. This work was published in [Zamora-Martínez *et al.*, 2010].

4. A preliminary NN LM based on Long-Short Term Memories, a kind of recurrent neural network, was presented. Promising results were obtained, but empirical evaluation suggests that IAM-DB handwritten text recognition task is not adequate to exploit this kind of recurrent neural network (lines to transcribe are composed of very few words). This work was performed in collaboration with members of the IAM of the University of Bern, and it was published in [Frinken *et al.*, 2012].

12.1.3. Contributions to Statistical Machine Translation

In this section the most relevant contribution is the adaptation of a two-level Viterbi algorithm to implement an Statistical Machine Translation decoding system. This adaptation is possible due to the generality of the developed algorithms and their implementation. Moreover, this algorithm integrates NN LMs into its core. The main contributions are:

1. Adaptation and extension of the two-level decoding algorithm of sequence recognition to work on machine translation problems using NN LMs. State-of-the-art results were obtained using this algorithm, compared to Moses. This work has been submitted to the COLING'2012 conference [Zamora-Martinez & Castro-Bleda, 2012].

2. Evaluation of NN LMs doing N-best-list rescoring over different SMT systems, obtaining competitive results [Khalilov *et al.*, 2008b,a; Zamora-Martínez & Sanchis, 2010; Zamora-Martinez & Castro-Bleda, 2011]. Coupling NN LMs into the decoding process improves in a non-statistically significant way the N-best list rescoring-based system [Zamora-Martinez *et al.*, 2010; Zamora-Martinez & Castro-Bleda, 2012]. Nevertheless, the computational cost is two or three times higher. A better adaptation of the Viterbi algorithm will be necessary to ensure better computational efficiency.

3. To solve the problem of different and large phrase-based translation tables combination, two approaches were evaluated: a linear interpolation smoothing, and a count-based smoothing. The two approaches obtained similar results, improving the baseline system in 0.4 absolute points of BLEU [Zamora-Martinez & Castro-Bleda, 2011].

4. Class-based NN LMs were trained to enhance the N-gram-based Translation Model when the bilingual tuple vocabulary was large and disperse. The obtained results show that, when the bilingual tuples vocabulary is huge, the class-based NN LMs approach improves a state-of-the-art system. This work has also been submitted to COLING'2012 [Zamora-Martinez & Castro-Bleda, 2012].

12.1.4. Summary

New extensions and contributions to connectionist language modeling were presented. Totally coupled decoding systems were also developed. The integration of NN LMs into the decoding process leads to a computational cost which is comparable to N-gram-based systems for sequence recognition tasks. For machine translation tasks, the computational cost was two or three times higher. Nevertheless, the quality of the developed NN LMs is very satisfactory, improving in every case the baseline in a statistically significant way. Extension to dynamic domain adaptation, developing the cache-based NN LMs, shows encouraging results, and motivates a deeper future research in this line.

12.2 Future work

Some tasks that can be improved using continuous space word projections for language modeling were found during this thesis writing, but they couldn't be developed inside its scope due to timing and length conditions. Nevertheless, without any doubt, these open research lines will be covered in the closest future. This section is a summary of these ideas.

12.2.1. Future work on connectionist language modeling

- A method to initialize word projections based on syntactic (POS tags) information was presented. Empirical results are needed to investigate its effect in the final model quality.

- The NN LM training algorithm using bunch mode, was modified to accept momentum term. The use of the momentum term increases the computational cost in $O(|\mathbf{W}|)$, but it is an important training parameter in order to achieve better convergence and stable results.
- Interpolation of standard LMs and NN LMs is non trivial. In this line, an idea that uses continuous space word projections to enhance this combination, following the Generalized Linear Interpolation (GLI) [Hsu, 2007], was presented. Some preliminary experiments showed the viability of this idea, motivating a future research on this line.

12.2.2. Future work on sequence recognition using NN LMs

- Better SLU schemes could be developed using confusion networks, decoupling the process in two steps: first an ASR step using improved models; and, second, a semantic tagging process on confusion networks. Continuous space and deep learning [Bengio, 2009] techniques could be used in order to improve the model quality.
- Out-of-vocabulary (OOV) words are crucial in sequence recognition tasks, being between a $3\,\%$ and $10\,\%$ of the running words, depending on the task. Character-based LMs could recover $\approx 33\,\%$ of this OOV words, giving an improvement of 1–3 absolute points of WER. In this way, an interesting future work will be to study a hybrid framework to combine character-based LMs inside a word-based decoder, in order to improve recognition results on OOV words.

12.2.3. Future work related with SMT

- Some of the models used in SMT tasks could be enhanced using the idea of continuous space projections:
  - The lexicalized reordering model could be estimated with neural networks, using a pair of bilingual tuples (or phrases) as input, and the output will be the probability of the orientation between the given pair.
  - The bilingual translation model could be enhanced adding to the NN LM input the previous context of tuples and its orientation.
  - The idea of Cache NN LMs could be applied to the target language NN LM, including in the cache not only the previous transcribed sentences, but the current source language sentence.
  - The training of a NN LM as a bilingual translation model suffers serious problems due to the size and dispersion of the vocabulary. The Structured Output Layer NN LM [Hai-Son *et al.*, 2011] could solve this problem.
- The combination of several phrase-tables trained with different domain corpora could be improved optimizing the weights using an automatic process.
- In the framework of the research projects HITITA[1] and STATE[2], an interactive transcription tool was developed for ancient handwritten document transcription. An inte-

[1] HITITA: Human-assisted interactive transcription of ancient printed and handwritten documents, `http://blogs.uji.es/hitita/`

[2] STATE: Sistema de transcripción asistida para texto escrito, `http://state.dlsi.uji.es/state/`

resting future line of research will be the integration of the SMT decoder developed in this thesis in order to obtain an interactive machine translation tool [Barrachina *et al.*, 2009].

12.3 Publications

12.3.1. 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

Currently this section does not contain any published work. Nevertheless, two papers were sent to indexed journals:

- Paper submitted to the "Pattern Recognition" journal [Zamora-Martínez *et al.*, 2012c]:

  F. Zamora-Martínez, V. Frinken, S. España-Boquera, M.J. Castro-Bleda, A. Fisher, H. Bunke. **Neural Network Language Models in Off-Line Handwriting Recognition**. *Pattern Recognition*.

- Paper submitted to the "IEEE Trans. on NNs" journal [Zamora-Martínez *et al.*, 2012b]:

  F. Zamora-Martínez, S. España-Boquera, M.J. Castro-Bleda. **Efficient integration of Neural Network Language Models for an HMM/ANN Off-Line Handwriting Recognition System**. *IEEE Transactions on Neural Networks*.

Publications in CORE indexed conferences

- **CORE A** indexed conference: submitted [Zamora-Martinez & Castro-Bleda, 2012]:

  F. Zamora-Martínez, M. J. Castro-Bleda. **Integrating Neural Network Language Models in the Decoding Stage of a Machine Translation System**. *COLING 2012*.

- **CORE A** indexed conference [Zamora-Martínez *et al.*, 2010]:

  F. Zamora-Martínez, M.J. Castro-Bleda, S. España-Boquera, J. Gorbe-Moya. **Unconstrained Offline Handwritting Recognition using Connectionist Character N-grams**. Pages 18–23 of the *International Joint Conference on Neural Networks*, Barcelona, 2010.

- **CORE B** indexed conference [Zamora-Martínez *et al.*, 2009]:

  F. Zamora-Martínez, M.J. Castro-Bleda, S. España-Boquera. **Fast Evaluation of Connectionist Language Models**. Pages 33–40 of "Bio-Inspired Systems: Computational and Ambient Intelligenc, Part I", volume 5517 of *Lecture Notes in Computer Science*, 2009. Salamanca, IWANN conference.

- **CORE B** indexed conference [Khalilov *et al.*, 2008b]:

  Maxim Khalilov, José A. R. Fonollosa, *F. Zamora-Martínez*, María J. Castro-Bleda, S. España-Boquera. **Neural Network Language Models for Translation with Limited Data**. Pages 445–451 of the *International Conference on Tools with Artificial Intelligence* proceedings, 2008, Dayton (USA).

Non indexed journals and conferences

Non indexed conferences but internationally recognized:

- International conference [Zamora-Martínez *et al.*, 2012a]:

  F. Zamora-Martínez, Salvador España-Boquera, M.J. Castro-Bleda, Renato de-Mori. **Cache Neural Network Language Models based on Long-Distance Dependencies for a Spoken Dialog System**. Pages 4993–4996 of the *IEEE International Conference on Acoustic, Speech and Signal Processing* proceedings, 2012, Kyoto (Japan).

Other journals and conferences:

- International workshop [Zamora-Martinez & Castro-Bleda, 2011]:

  F. Zamora-Martínez, M.J. Castro-Bleda. **CEU-UPV English-Spanish system for WMT11**. Pages 490–495 of the *6th Workshop on Statistical Machine Translation and MetricsMATR* proceedings, 2011, Edinburgh (Scotland).

- International workshop [Zamora-Martinez *et al.*, 2010]:

  F. Zamora-Martínez, M.J. Castro-Bleda, H. Schwenk. **Ngram-based Machine Translation enhanced with Neural Networks for the French-English BTEC-IWSLT'10 task.**. Pages 45–52 of the *seventh International Workshop on Spoken Language Translation (IWSLT)* proceedings. 2010, Paris (France).

- International workshop [Zamora-Martínez & Sanchis, 2010]:

  F. Zamora-Martínez, Germán Sanchis-Trilles. **UCH-UPV English-Spanish system for WMT10**. Pages 207–211 of the *5th Workshop on Statistical Machine Translation and MetricsMATR* proceedings, 2010, Uppsala (Sweden).

- National journal [Zamora-Martínez & Castro-Bleda, 2010]:

  F. Zamora-Martínez, M.J. Castro-Bleda. **Traducción Automática Estadística basada en Ngramas Conexionistas**. Pages 221–228 of the *Sociedad Española para el Procesamiento del Lenguaje Natural* journal. Volume 45, number 45, 2010. Valencia (Spain).

- International workshop [Khalilov *et al.*, 2008a]:

Maxim Khalilov, José A. R. Fonollosa, *F. Zamora-Martínez*, María J. Castro-Bleda, S. España-Boquera. **Arabic-English translation improvement by target-side neural network language modeling**. In proceedings of *Workshop HLT & NLP within the Arabic world* at LREC, 2008, Marrakech (Marocco).

- International conference [España-Boquera *et al.*, 2007a]:

  S. España-Boquera, *F. Zamora-Martínez*, M.J. Castro-Bleda, J. Gorbe-Moya. **Efficient BP Algorithms for General Feedforward Neural Networks**. Pages 327–336 de "Bio-inspired Modeling of Cognitive Tasks, Part I", volume 4527 of *Lecture Notes in Computer Science*, 2007. Murcia, IWINAC conference.

- National conference [Zamora-Martínez *et al.*, 2006]:

  F. Zamora-Martínez, Salvador España, Jorge Gorbe, María José Castro. **Entrenamiento de modelos de lenguaje conexionistas con grandes vocabularios**. Pages 141–144 of the *IV Jornadas de Tecnología del Habla*, 2006, Zaragoza.

12.3.2. Collaborations with other authors

This section contains publications that are important for this thesis under the role of a collaboration author. Some of these contributions describe tools and algorithms important for the experimental framework.

Publications in JCR indexed journals

- JCR indexed journal, score 5.308, first quart (1 of 108) [España-Boquera *et al.*, 2011], from the thesis of Salvador España [España-Boquera, 2012]:

  S. España-Boquera, M.J. Castro-Bleda, J. Gorbe-Moya, *F. Zamora-Martínez*. **Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models**. Pages 767–779 of the *IEEE Transactions on Pattern Analysis and Machine Intelligence* journal. Volume 33, number 4, 2011.

Publications in CORE indexed conferences

- **CORE B** international conference [Frinken *et al.*, 2012]:

  Volkmar Frinken, *F. Zamora-Martínez*, Salvador España-Boquera, María J. Castro-Bleda, Andreas Fischer, Horst. Bunke. **Long-Short Term Memory Neural Networks Language Modeling for Handwriting Recognition**. Accepted for publication in *International Conference on Pattern Recognition* proceedings, 2012, Tsukuba (Japan).

Non-indexed conferences and journals

- National conference [Zamora-Martínez *et al.*, 2010]:

  F. Zamora-Martínez, M.J. Castro-Bleda, S. España-Boquera, J. Gorbe-Moya. **Improving Isolated Handwritten Word Recognition Using a Specialized Classifier for Short Words**. Pages 61–70 of "Conferencia de la Asociación Española para la Inteligencia Artificial" proceedings, volume 5988 of *Lecture Notes on Artificial Intelligence*. Sevilla, CAEPIA 2009 conference.

- International conference [Gorbe *et al.*, 2008]:

  J. Gorbe-Moya, S. España-Boquera, *F. Zamora-Martínez*, M.J. Castro-Bleda. **Handwritten Text Normalization by using Local Extrema Classification**. Pages 164–172 of *Pattern Recognition in Information Systems* workshop, Barcelona, 2008.

- International conference [España-Boquera *et al.*, 2007b]:

  S. España-Boquera, M.J. Castro-Bleda, *F. Zamora-Martínez*, J. Gorbe-Moya. **Efficient Viterbi Algorithms for Lexical Tree Based Models**. Pages 179–187 de "International Conference on Advances in Nonlinear Speech Processing", volume 4885 of *Lectures Notes on Artificial Intelligence*, Paris, 2007.

- International conference [España-Boquera *et al.*, 2007]:

  S. España-Boquera, J. Gorbe-Moya, *F. Zamora-Martínez*. **Semiring Lattice Parsing Applied to CYK**. Pages 603–610 de "Pattern Recognition and Image Analysis", volume 4477 of *Lecture Notes on Computer Science*. Girona, IBPRIA conference, 2007.

12.3.3. Other publications

During these years some contributions in other related research fields, out of the main focus of this thesis, were done. Some of them are about Part-of-Speech tagging, or text image cleaning and enhancing. Group collaborations in the framework of research projects are also included.

- **CORE B** international conference [Castro-Bleda *et al.*, 2011]:

  M.J. Castro-Bleda, S. España-Boquera, David Llorens, Andrés Marzal, Federico Prat, Juan Miguel Vilar, *F. Zamora-Martínez*. **Speech interaction in a multimodal tool for handwritten text transcription**. Pages 299–302 of the *International Conference on Multimodal Interaction*, 2011, Alicante (Spain).

- **CORE B** international conference [España Boquera *et al.*, 2010]:

S. España-Boquera, J. Gorbe-Moya, *F. Zamora-Martínez* and M. J. Castro-Bleda. **Hybrid HMM/ANN models for bimodal online and offline cursive word recognition**. Pages 299–302 of the *International Conference on Pattern Recognition*, 2010, Istanbul (Turkey).

- **CORE A** international conference [Castro *et al.*, 2009]:

  M.J. Castro, S. España, J. Gorbe, *F. Zamora-Martínez*, D. Llorens, A. Marzal, F. Prat, J.M. Vilar **Improving a DTW-based recognition engine for on-line handwritten characters by using MLPs**. Pages 1260–1264 of the *International Conference on Document Analysis and Recognition*. 2009, Barcelona.

- International conference [Zamora-Martínez *et al.*, 2009a]:

  F. Zamora-Martínez, M.J. Castro-Bleda, S. Tortajada-Velert, S. España-Boquera. **A Connectionist approach to Part-Of-Speech Tagging**. Pages 421–426 of the *International Conference on Neural Computation*, 2009, Funchal (Portugal).

- National conference [Zamora-Martínez *et al.*, 2009b]:

  F. Zamora-Martínez, M.J. Castro-Bleda, S. Tortajada-Velert, S. España-Boquera. **Adding morphological information to a connectionist Part-Of-Speech tagger**. Pages 169–178 of the *Conferencia de la Asociación Española para la Inteligencia Artificial*, 2009, Sevilla.

- National conference [Segarra *et al.*, 2008]:

  E, Segarra, M.J. Castro, I. Galiano, F. García, J.A. Gómez, D. Griol, L.F. Hurtado, E, Sanchis, F. Torres, *F. Zamora-Martínez*. **Adquisición de un corpus de diálogos para un dominio de reservas de instalaciones deportivas**. Pages 163–166 of the *V Jornadas en Tecnología del Habla* proceedings. 2008, Bilbao.

- International conference [Llorens *et al.*, 2008]:

  D. Llorens, F. Prat, A. Marzal, J.M. Vilar, M.J. Castro, J.C. Amengual, S. Barrachina, A. Castellanos, S. España, J.A. Gómez, J. Gorbe, A. Gordo, V. Palazón, G. Peris, R. Ramos-Garijo, *F. Zamora-Martínez*. **The UJIpenchars Database: A Pen-Based Database of Isolated Handwritten Characters**. In proceedings of the *6th International Conference on Language Resources and Evaluation (LREC 2008)*. Marrakech, Marocco.

- **CORE B** international conference [Zamora-Martínez *et al.*, 2007]:

  F. Zamora-Martínez, Salvador España, M.J. Castro. **Behaviour-based Clustering of Neural Networks applied to Document Enhancement**. Pages 144–151 of "Computational and Ambient Intelligence", volume 4507 of *Lecture Notes in Computer Science*. San Sebastián, IWANN conference, 2007.

BIBLIOGRAPHY

Afify, M., Siohan, O., & Sarikaya, R. 2007. Gaussian mixture language models for speech recognition. *Pages 29–32 of: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. (Cited on page 86).

Banchs, R., Crego, J., Gispert, A., Lambert, P., Fonollosa, J. A., Costa-jussà, M. R., & Mariño, J. 2005. Bilingual N-gram Statistical Machine Translation. *Pages 275–282 of: Proceedings of the MT Summit*. (Cited on pages 166 and 311).

Bangalore, S., & Riccardi, G. 2000. Stochastic finite-state models for spoken language machine translation. *Pages 52–59 of: Proceedings of the Workshop on Embedded machine translation systems*. (Cited on pages 166 and 176).

Barrachina, S., Bender, O., Casacuberta, F., Civera, J., Cubel, E., Khadivi, S., Lagarda, A., Ney, H., Tomás, J., Vidal, E., & Vilar, J.-M. 2009. Statistical approaches to computer-assisted translation. *Computational Linguistics*, **35**(1), 3–28. (Cited on page 241).

Bauer, L. 1993. *Manual of Information to Accompany The Wellington Corpus of Written New Zealand English*. Tech. rept. Victoria University, Department of Linguistics. (Cited on page 137).

Baum, L. E., & Egon, J. A. 1967. An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology. *Annals of Mathematical Statistics*, **73**, 360–363. (Cited on page 97).

Baum, L. E., & Sell, G. R. 1968. Growth functions for transformations on manifolds. *Pacific Journal of Mathematics*, **27**(2), 211–227. (Cited on page 97).

Bellegarda, J. R. 2000. Large vocabulary speech recognition with multispan statistical language models. *IEEE Transactions on Speech and Audio Processing*, **8**(1), 76 –84. (Cited on page 86).

Bengio, Y. 2008. Neural net language models. *Scholarpedia*, **3**(1), 3881. (Cited on pages 5, 27, 29, 237, 289 and 293).

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. 2001. A Neural Probabilistic Language Model. *Pages 933–938 of: Proceedings of the Advances in Neural Information Processing Systems Conference*. (Cited on pages 5 and 27).

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, **3**(2), 1137–1155. (Cited on pages 5, 27, 29, 35, 43, 289, 293 and 294).

Bengio, Y. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, **2**(1). (Cited on page 240).

Bengio, Y., & Sénécal, J.-S. 2008. Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model. *IEEE Transactions on Neural Networks*, **19**(4), 713–722. (Cited on pages 68 and 69).

Bengio, Y., LeCun, Y., Nohl, C., & Burges, C. 1995. LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition. *Neural Computation*, **7**(6), 1289–1303. (Cited on page 133).

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. 2006. Greedy Layer-Wise Training of Deep Networks. *Pages 153–160 of: Proceedings of the Advances in Neural Information Processing Systems Conference*. (Cited on page 42).

Berger, A., Pietra, S. D., & Pietra, V. D. 1996a. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, **22**(1), 39–72. (Cited on pages 8 and 290).

Berger, A. L., Brown, P. F., Pietra, S. A. D., & Pietra, V. D. 1996b. *Language Translation Apparatus and Method of Using Context-based Translation Models*. United States Patent 5510981. (Cited on pages 187 and 314).

Berry, M. W., Dumais, S. T., & O'Brien, G. W. 1995. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, **37**(4), 573–595. (Cited on page 86).

Bertolami, R., & Bunke, H. 2008a. Ensemble Methods to Improve the Performance of an English Handwritten Text Line Recognizer. *Pages 265–277 of: Arabic and Chinese Handwriting Recognition*. Lecture Notes in Computer Science, vol. 4768. Springer. (Cited on page 134).

Bertolami, R., & Bunke, H. 2008b. Hidden Markov models-based ensemble methods for offline handwritten text line recognition. *Pattern Recognition*, **41**(11), 3452–3460. (Cited on page 134).

Bishop, C. M. 1995. *Neural networks for pattern recognition*. Oxford University Press. (Cited on pages 36, 67, 98, 276, 295 and 305).

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer-Verlag. (Cited on page 62).

BLAS. 1979. `http://www.netlib.org/blas/`. (Cited on pages 50 and 298).

Blei, D. M., & Jordan, M. I. 2002. Modeling annotated data. *In: Technical Report*. (Cited on page 86).

Bonneau-Maynard, H., *et al.* 2005. Semantic Annotation of the French Media Dialog Corpus. *Pages 3457–3460 of: Proceedings of the annual Conference of the International Speech Communication Association*. (Cited on pages 29, 113 and 293).

Bourlard, H., & Morgan, N. 1994. *Connectionist speech recognition—A hybrid approach*. Engineering and computer science, vol. 247. Kluwer Academic. (Cited on pages 98, 99, 116 and 305).

Bozinovic, R. M., & Srihari, S. N. 1989. Off-Line Cursive Script Word Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**(1), 68–83. (Cited on page 132).

Bridle, J. S. 1990. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Pages 211–217 of: Proceedings of the Advances in Neural Information Processing Systems Conference*. (Cited on page 98).

Brown, P., Cocke, J., Pietra, S. D., Pietra, V. D., Jelinek, F., Lafferty, J. D., Mercer, R., & Roossin, P. S. 1990. A statistical approach to machine translation. *Computational Linguistics*, **16**(2), 79–85. (Cited on pages 163 and 310).

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., & Mercer, R. L. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, **19**(2), 263–311. (Cited on pages 166, 170 and 312).

Brown, P., Della Pietra, S. A., Della Pietra, J. D., Lai, J. C., & Mercer, R. L. 1992. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, **18**(1), 31–40. (Cited on pages 14, 32, 290 and 294).

Burr, D. J. 1982. A normalizing transform for cursive script recognition. *Pages 1027–1030 of: Proceedings of the International Conference on Pattern Recognition*. (Cited on page 132).

Casacuberta, F., & Vidal, E. 2004. Machine Translation with Inferred Stochastic Finite-State Transducers. *Computational Linguistics*, **30**, 205–225. (Cited on pages 166, 167, 311, 312 and 313).

Castro, M. J., España, S., Gorbe, J., Zamora-Martínez, F., Llorens, D., Marzal, A., Prat, F., & Vilar, J. M. 2009. Improving a DTW-based recognition engine for on-line handwritten characters by using MLPs. *Pages 1260–1264 of: Proceedings of the International Conference on Document Analysis and Recognition*, vol. 5517. (Cited on page 245).

Castro-Bleda, M. J., & Prat, F. 2003. New Directions in Connectionist Language Modeling. *Pages 598–605 of: Computational Methods in Neural Modeling*. Lecture Notes in Computer Science, vol. 2686. Springer-Verlag. (Cited on pages 5, 27 and 289).

Castro-Bleda, M. J., Prat, F., & Casacuberta, F. 1999. MLP emulation of N-gram models as a first step to connectionist language modeling. *Pages 910–915 of: Proceedings of the International Conference of Artificial Neural Networks*, vol. 2. (Cited on pages 35 and 294).

Castro-Bleda, M. J., España-Boquera, S., Llorens, D., Marzal, A., Prat, F., Vilar, J. M., & Zamora-Martínez, F. 2011. Speech interaction in a multimodal tool for handwritten text transcription. *Pages 299–302 of: Proceedings of the International Conference on Multimodal Interaction*. (Cited on page 244).

Chen, S., Beeferman, D., & Rosenfeld, R. 1998. Evaluation Metrics For Language Models. *In: Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*. (Cited on pages 28 and 293).

Chen, S. F., & Goodman, J. 1996. An Empirical Study of Smoothing Techniques for Language Modeling. *Pages 310–318 of: Proceedings of the annual meeting of the Association for Computational Linguistics*. (Cited on pages 29 and 293).

Chien, J. T., & Chueh, C. H. 2011. Dirichlet Class Language Models for Speech Recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, **19**(3), 482–495. (Cited on pages 85 and 86).

Clarkson, P., & Robinson, T. 1999. Towards Improved Language Model Evaluation Measures. *Page pages of: Proceedings of the European Conference on Speech Communications and Technology*. (Cited on pages 28 and 293).

Coccaro, N., & Jurafsky, D. 1998. Towards Better Integration Of Semantic Predictors In Statistical Language Modeling. *Pages 2403–2406 of: Proceedings of the International Conference on Spoken Language Processing*. (Cited on page 86).

Costa-jussà, M. R., & Fonollosa, J. A. R. 2006. Statistical machine reordering. *Pages 70–76 of: Proceedings of the Empirical Methods in Natural Language Processing Conference*. (Cited on pages 166, 311 and 313).

Crego, J., de Gispert, A., Lambert, P., Khalilov, M., Costa-jussà, M., Mariño, J., Banchs, R., & Fonollosa, J. A. 2006a. The TALP Ngram-based SMT System for IWSLT 2006. *Pages 116–122 of: Proceedings of the International Workshop on Spoken Language Translation*. (Cited on page 210).

Crego, J. M., Mariño, J., & de Gispert, A. 2005a. An Ngram-based Statistical Machine Translation Decoder. *Pages 3185–3188 of: Proceedings of the annual Conference of the International Speech Communication Association*. (Cited on pages 210 and 316).

Crego, J. M., de Gispert, A., & Mariño, J. M. 2005b. MARIE, Ngram-based Statistical Machine Translation decoder. *Pages 3193–3196 of: Proceedings of the annual Conference of the International Speech Communication Association*. (Cited on pages 209 and 316).

Crego, J. M., Mariño, J. B., & de Gispert, A. 2005c. Reordered search and tuple unfolding for ngram-based SMT. *In: Proceedings of the MT Summit*. (Cited on pages 175 and 184).

Crego, J. M., Costa-jussà, M. R., Mariño, J. B., de Gispert, A., Khalilov, M., Fonollosa, J. A. R., Lambert, P., & Banchs, R. E. 2006b. N-gram-based SMT system enhanced with reordering patterns. *Pages 162–165 of: Proceedings of the Workshop on Statistical Machine Translation*. (Cited on pages 175, 184 and 187).

Dempster, A. P., Laird, N. M., & Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc.*, **39**(1), 1–38. (Cited on page 97).

Duda, R. O., Hart, P. E., & Stork, D. G. 2001. *Pattern Classification*. Second edn. John Wiley and Sons. (Cited on pages 8, 40 and 289).

Elman, J. L. 1990. Finding structure in time. *Cognitive Science*, **14**, 179–211. (Cited on pages 35 and 294).

Emami, A., & Mangu, L. 2007. Empirical study of Neural Network Language Models for Arabic Speech Recognition. *Pages 147–152 of: Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, vol. 7. (Cited on pages 5, 49, 289 and 298).

Eppstein, D. 1998. Finding the k shortest paths. *SIAM J. Computing*, **28**(2), 652–673. (Cited on pages 106, 181 and 313).

España Boquera, S., Gorbe-Moya, J., Zamora-Martínez, F., & Castro-Bleda, M. J. 2010. Hybrid HMM/ANN models for bimodal online and offline cursive word recognition. *Pages 14–21 of: Proceedings of the International Conference on Pattern Recognition*. (Cited on page 244).

España-Boquera, S., Gorbe-Moya, J., & Zamora-Martínez, F. 2007. Semiring Lattice Parsing Applied to CYK. *Pages 603–610 of: Pattern Recognition and Image Analysis*. Lecture Notes in Computer Science, vol. 4477. (Cited on pages 10, 191, 244, 290 and 304).

España-Boquera, S., Zamora-Martínez, F., Castro-Bleda, M. J., & Gorbe-Moya, J. 2007a. Efficient BP Algorithms for General Feedforward Neural Networks. *Pages 327–336 of: Bio-inspired Modeling of Cognitive Tasks*. Lecture Notes in Computer Science, vol. 4527. Springer. (Cited on pages 48, 243 and 298).

España-Boquera, S., Castro-Bleda, M. J., Zamora-Martínez, F., & Gorbe-Moya, J. 2007b. Efficient Viterbi algorithms for lexical tree based models. *Pages 179–187 of: Proceedings of the International Conference on Non-Linear Speech Processing*. Lecture Notes in Computer Science, vol. 4885. Springer. (Cited on pages 102, 244, 304 and 305).

España-Boquera, S., Castro-Bleda, M. J., Gorbe-Moya, J., & Zamora-Martinez, F. 2011. Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **33**(4), 767–779. (Cited on pages XXXIII, XXXIII, XXXIV, XXXIV, 97, 128, 130, 131, 133, 134, 154, 243, 304, 305 and 308).

España-Boquera, S. 2012. *Contributions to the joint classification and segmentation of sequences*. PhD Thesis on Computer Science, Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València. Advisors: Castro-Bleda, M. J. and Marzal-Varó, Andrés. (Cited on pages XV, 99, 106 and 243).

Fiscus, J. 1997. A Post-Processing System to Yield Reduced Word Error Rates: Recognizer Output Voting Error Reduction (ROVER). *Pages 347–354 of: Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*. (Cited on page 155).

Florian, R., & Yarowsky, D. 1999. Dynamic nonlocal language modeling via hierarchical topic-based adaptation. *Pages 167–174 of: Proceedings of the annual meeting of the Association for Computational Linguistics*. (Cited on page 85).

Foster, G., Goutte, C., & Kuhn, R. 2010. Discriminative instance weighting for domain adaptation in statistical machine translation. *Pages 451–459 of: Proceedings of the Empirical Methods in Natural Language Processing Conference*. (Cited on page 214).

Francis, W. N., & Kucera, H. 1979. *Brown Corpus Manual, Manual of Information to accompany A Standard Corpus of Present-Day Edited American English*. Technical Report. Brown University, Department of Linguistics. (Cited on page 137).

Frinken, V., Zamora-Martínez, F., España-Boquera, S., Castro-Bleda, M. J., & Fisher, A. 2012. Long-Short Term Memory Neural Networks Language Modeling for Handwriting Recognition. *In: Proceedings of the International Conference on Pattern Recognition*. (Cited on pages XXXIV, XXXIV, XXXIV, 44, 128, 151, 152, 153, 155, 238, 243, 308, 309 and 310).

Galliano, S., *et al.* 2005. The ESTER Phase II Evaluation Campaign for the Rich Transcription of French Broadcast New. *In: Proceedings of the annual Conference of the International Speech Communication Association*. (Cited on page 113).

Good, I. J. 1953. The poluation frequencies of species and the estimation of population parameters. *Biometrika*, **40**(3 and 4), 237–264. (Cited on pages 31 and 294).

Gorbe, J., España, S., Zamora-Martínez, F., & Castro, M. J. 2008. Handwritten Text Normalization by using Local Extrema Classification. *Pages 164–172 of: Proceedings of the International Workshop on Pattern Recognition in Information Systems*. (Cited on pages 130, 132, 133 and 244).

Graves, A., Fernandez, S., Liwicki, M., Bunke, H., & Schmidhuber, J. 2008. Unconstrained online handwriting recognition with recurrent neural networks. *Pages 577–584 of: Proceedings of the Advances in Neural Information Processing Systems Conference*. (Cited on page 151).

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., & Schmidhuber, J. 2009. A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **31**(5), 855–868. (Cited on pages 134, 140, 142, 143, 145, 151, 153 and 308).

Hahn, S., Dinarelli, M., Raymond, C., Lefevre, F., Lehnen, P., De Mori, R., Moschitti, A., Ney, H., & Riccardi, G. 2011. Comparing Stochastic Approaches to Spoken Language Understanding in Multiple Languages. *IEEE Transactions on Audio, Speech, and Language Processing*, **19**(6), 1569–1583. (Cited on pages XXVI, XXVI, 120, 123, 124 and 307).

Hai-Son, L., Alluzen, A., Wisniewski, G., & Yvon, F. 2010. Training Continuous Space Language Models: Some Practical Issues. *Pages 778–788 of: Proceedings of the Empirical Methods in Natural Language Processing Conference*. (Cited on pages 42, 43, 47, 296 and 297).

Hai-Son, L., Oparin, I., Alluzen, A., Gauvain, J.-L., & Yvon, F. 2011. Structured Output Layer Neural Network Language Model. *Pages 5524–5527 of: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 11. (Cited on pages 43, 50, 67, 68, 226, 231, 240 and 299).

Hidalgo, J. L., España, S., Castro, M. J., & Pérez, J. A. 2005. Enhancement and cleaning of handwritten data by using neural networks. *Pages 376–383 of: Pattern Recognition and Image Analysis*. Lecture Notes in Computer Science, vol. 3522. Springer-Verlag. (Cited on page 132).

Hinton, G. 1986. Learning distributed representations of concepts. *Pages 1–12 of: Proceedings of the annual Conference of the Cognitive Science Society*. (Cited on pages 35 and 294).

Hinton, G. 2000. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, **14**, 2002. (Cited on page 42).

Hochreiter, S., & Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*, **9**(8), 1735–1780. (Cited on page 151).

Hofmann, T. 1998. *Unsupervised Learning from Dyadic Data*. MIT Press. (Cited on page 86).

Hsu, B. 2007. Generalized linear interpolation of language models. *Pages 136–140 of: Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*. (Cited on pages 62, 240 and 301).

Ito, A., Kohda, M., & Ostendorf, M. 1999. A new metric for stochastic language model evaluation. *Pages 1591–1594 of: Proceedings of the European Conference on Speech Communications and Technology*. (Cited on pages 28 and 293).

Iyer, R., Ostendorf, M., & Meteer, M. 1997. *Analyzing and Predicting Language Model Improvements*. (Cited on pages 28 and 293).

Jeffreys, H. 1948. *Theory of Probability*. Clarendon Press, Oxford. (Cited on pages 31 and 294).

Jelinek, F. 1997. *Statistical Methods for Speech Recognition*. Language, Speech, and Communication. The MIT Press. (Cited on pages 27, 29 and 293).

Jelinek, F., & Mercer, R. L. 1980. Interpolated estimation of Markov source parameters from sparse data. *In: Proceedings of the Workshop on Pattern Recognition in Practice*. (Cited on pages 32 and 294).

Jiménez, V. M., & Marzal, A. 2003. A lazy version of eppstein's k shortest paths algorithm. *Pages 179–190 of: Proceedings of the International Conference on Experimental and efficient algorithms*. Springer. (Cited on pages 106, 181 and 313).

Johansson, S., Leech, G. N., & Goodluck, H. 1978. *Manual of Information to accompany the Lancaster-Oslo/Bergen Corpus of British English, for use with digital Computers*. Technical Report. University of Oslo, Department of English. (Cited on page 73).

Johansson, S., Atwell, E., Garside, R., & Leech, G. 1986. *The Tagged LOB Corpus: User's Manual*. Norwegian Computing Centre for the Humanities. (Cited on pages 134 and 137).

Johnson, W. E. 1932. Probability: deductive and inductive problems. *Mind*, **41**, 421–423. (Cited on pages 31 and 294).

Jurafsky, D., & Martin, J. H. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall. (Cited on pages XXXIV and 162).

Kanthak, S., Vilar, D., Matusov, E., Zens, R., & Ney, H. 2005. Novel reordering approaches in phrase-based statistical machine translation. *Pages 167–174 of: Workshop on Building and Using Parallel Texts: Data-Driven Machine Translation and Beyond.* (Cited on page 187).

Khalilov, M., Fonollosa, J. A. R., Zamora-Martínez, F., Castro, M. J., & España, S. 2008a. Arabic-English translation improvement by target-side neural network language modeling. *In: Proceedings of the Workshop HLT & NLP within the Arabic world*. (Cited on pages 209, 232, 239, 242 and 316).

Khalilov, M., Fonollosa, J. A. R., Zamora-Martínez, F., Castro-Bleda, M. J., & España-Boquera, S. 2008b. Neural Network Language Models for Translation with Limited Data. *Pages 445–451 of: Proceedings of the International Conference on Tools with Artificial Intelligence*. (Cited on pages 209, 210, 232, 239, 242 and 316).

Khalilov, M., Zamora-Martinez, F., Fonollosa, J. A., Castro-Bleda, M. J., & España-Boquera, S. 2012. *Selecting the Best Translation in the Last Step with Neural Networks*. Technical report. Universitat Politècnica de València, Departamento de Sistemas Informáticos y Computación. Preparing. (Cited on pages 210 and 316).

Klakow, D., & Peters, J. 2002. Testing the correlation of word error rate and perplexity. *Speech Commun.*, **38**(1), 19–28. (Cited on pages 14, 28 and 293).

Kneser, R., & Ney, H. 1995. Improving backing-off for m-gram language modeling. *Pages 181–184 of: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. (Cited on pages 33, 34, 213 and 294).

Knight, K., & Al-Onaizan, Y. 1998. Translation with Finite-State Devices. *Pages 421–437 of: Proceedings of the Association for Machine Translation in the Americas Conference*. (Cited on page 176).

Koehn, P. 2004. Statistical significance tests for machine translation evaluation. *Pages 388–395 of: Proceedings of the Empirical Methods in Natural Language Processing Conference*. (Cited on pages 17, 213 and 291).

Koehn, P., Och, F. J., & Marcu, D. 2003. Statistical phrase-based translation. *Pages 48–54 of: Proceedings of the annual meeting of the Nord American chapter of the Association for Computational Linguistics on Human Language Technologies*. (Cited on pages 164 and 310).

Koehn et al., P. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. *Pages 177–180 of: Proceedings of the ACL Demo and Poster Sessions*. (Cited on pages 19, 173, 177, 189, 209, 210, 212, 291, 312, 314 and 316).

Kuhn, R., & Mori, R. D. 1990. A Cache-Based Natural Language Model for Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**, 570–583. (Cited on pages 46, 58, 83, 85, 300 and 304).

Lafferty, J., *et al.* 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Pages 282–289 of: Proceedings of the Machine Learning Conference*. (Cited on page 119).

Landauer, T. K., & Dumais, S. T. 1997. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*. (Cited on page 86).

Lawson, C. L., Hanson, R. J., Kincaid, D., & Krogh, F. T. 1979. Basic Linear Algebra Subprogramas for FORTRAN usage. *ACM Transactions on Mathematical Software*, **5**, 308–323. (Cited on pages 50, 281 and 298).

Legetter, C. J., & Woodland, P. C. 1995. Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech and Language*, **9**, 171–185. (Cited on page 86).

Lidstone, G. J. 1920. Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, **8**, 182–192. (Cited on pages 31 and 294).

Linarès, G., *et al.* 2007. The LIA speech recognition system: from 10xRT to 1xRT. *Pages 302–308 of: Proceedings of the International Conference on Text, Speech and Dialogue*. (Cited on page 113).

Llorens, D., Prat, F., Marzal, A., Vilar, J. M., Castro, M. J., Amengual, J. C., Barrachina, S., Castellanos, A., España, S., Gómez, J. A., Gorbe, J., Gordo, A., Palazón, V., Peris, G., Ramos-Garijo, R., & Zamora-Martínez, F. 2008. The UJIpenchars Database: A Pen-Based Database of Isolated Handwritten Characters. *In: Proceedings of the Conference on Language Resorces and Evaluation*. (Cited on page 245).

Llorens Piñana, D. 2000. *Suavizado de automatas y traductores finitos estocasticos.* PhD Thesis on Computer Science, Universitat Politècnica de València. (Cited on pages 53, 168 and 299).

Mariño, J. B., Banchs, R. E., Crego, J. M., de Gispert, A., Lambert, P., Fonollosa, J. A. R., & Costa-jussà, M. R. 2006. N-gram-based Machine Translation. *Computational Linguistics*, **32**, 527–549. (Cited on pages 165, 166, 170, 176, 311 and 312).

Marti, U., & Bunke, H. 1999. A full English sentence database for off-line handwriting recognition. *Pages 705–708 of: Proceedings of the International Conference on Image Analysis and Recognition*. (Cited on pages 29 and 293).

Marti, U. V., & Bunke, H. 2002. The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, **5**, 39–46. (Cited on page 134).

Marzal, A. 2006. *Apuntes de Reconocimiento Automático del Habla*. (Cited on pages XXXIII, XXXIII, XXXIII, 109, 110, 112 and 113).

Matsoukas, S., Rosti, A.-V. I., & Zhang, B. 2009. Discriminative corpus weight estimation for machine translation. *Pages 708–717 of: Proceedings of the Empirical Methods in Natural Language Processing Conference*, vol. 2. (Cited on page 214).

Matsumoto, M., & Nishimura, T. 1998. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, **8**(1), 3–30. (Cited on page 282).

Matusov, E., Kanthak, S., & Ney, H. 2005. Efficient Statistical Machine Translation with Constrained Reordering. *Pages 181–188 of: Proceedings of the International Conference of the European Association for Machine Translation*. (Cited on pages 187 and 314).

Miikkulainen, R., & Dyer, M. G. 1991. Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, **15**, 343—399. (Cited on pages 35 and 294).

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. 2010. Recurrent neural network based language model. *Pages 1045–1048 of: Proceedings of the annual Conference of the International Speech Communication Association*. (Cited on pages 43, 67 and 151).

Mikolov, T., Kombrink, S., Burget, L., Cernocký, J., & Khudanpur, S. 2011. Extensions of recurrent neural network language model. *Pages 5528–5531 of: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. (Cited on pages 43 and 151).

Mitchell, J., & Lapata, M. 2009. Language Models Based on Semantic Composition. *Pages 430—-439 of: Proceedings of the Empirical Methods in Natural Language Processing Conference*. (Cited on page 86).

Mnih, A., & Hinton, G. 2007. Three new graphical models for statistical language modelling. *Pages 641–648 of: Proceedings of the International Conference on Machine learning*. (Cited on pages 42 and 43).

Mnih, A., & Hinton, G. 2008. A Scalable Hierarchical Distributed Language Model. *Pages 1–8 of: Proceedings of the Advances in Neural Information Processing Systems Conference*. (Cited on page 43).

Morin, F., & Bengio, Y. 2005. Hierarchical probabilistic neural network language model. *Pages 246–252 of: Proceedings of the International Conference on Artificial Intelligence and Statistics*. (Cited on pages 5, 50, 68, 289 and 299).

Nadeu, C., *et al.* 1996. Frequency and time filtering of filter-bank energies for HMM speech recognition. *Pages 430–433 of: Proceedings of the International Conference on Spoken Language Processing*, vol. 1. (Cited on pages 112 and 113).

Nakamura, M., & Shikano, K. 1989. A study of English word category prediction based on neural networks. *Pages 731–734 of: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. (Cited on pages 35 and 294).

Nelder, J. A., & Mead, R. 1965. A simplex method for function minimization. *The Computer organization*, **7**, 308–313. (Cited on pages 19, 97, 211, 291 and 305).

Ney, H., & Essen, U. 1991. On Smoothing Techniques for Bigram-Based Natural Language Modeling. *Pages 825–828 of: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. (Cited on pages 29 and 293).

Och, F., & Ney, H. 2002. Discriminative training and maximum entropy models for statistical machine translation. *Pages 295–302 of: Proceedings of the annual meeting of the Association for Computational Linguistics*. (Cited on pages 165 and 310).

Och, F. J. 2003. Minimum Error Rate Training in Statistical Machine Translation. *Pages 160–167 of: Proceedings of the annual meeting of the Association for Computational Linguistics*. (Cited on pages 19 and 291).

Och, F. 1999. An Efficient Method for Determining Bilingual Word Classes. *Pages 71–76 of: Proceedings of the annual meeting of the European chapter of the Association for Computational Linguistics*. (Cited on page 226).

Och, F. J., & Ney, H. 2003. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, **29**(1), 19–51. (Cited on pages 164, 166, 168, 214, 220, 224, 226 and 310).

Oparin, I., Sundermeyer, M., Ney, H., & Gauvain, J.-L. 2012. Performance analysis of Neural Networks in combination with N-Gram Language Models. *In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. (Cited on page 60).

Paccanaro, A., & Hinton, G. 2000. Extracting distributed representations of concepts and relations from positive and negative propositions. *Pages 2259—2264 of: Proceedings of the International Joint Conference on Neural Networks*. (Cited on pages 35 and 294).

Palacios, A. 2012. *Implementación de un módulo para el entrenamiento y evaluación de redes neuronales mediante GPUs*. Degree final project on Computer Science, Escuela Técnica Superior de Informática, Universitat Politècnica de València. Advisors: S. España, F. Zamora-Martínez. (Cited on page 50).

Papineni, K., Roukos, S., Ward, T., & Zhu, W. 2002. Bleu: a method for automatic evaluation of machine translation. *Pages 311–318 of: Proceedings of the annual meeting of the Association for Computational Linguistics*. (Cited on pages 15 and 291).

Park, J., Liu, X., Gales, M. J., & Woodland, P. C. 2010. Improved Neural Network Based Language Modelling and Adaptation. *Pages 26–30 of: Proceedings of the annual Conference of the International Speech Communication Association*. (Cited on pages 5, 49, 289 and 298).

Pastor, M., Toselli, A., & Vidal, E. 2004. Projection profile based algorithm for slant removal. *Pages 183–190 of: Proceedings of the International Conference on Image Analysis and Recognition*. Lecture Notes in Computer Science, vol. 3212. (Cited on page 132).

Powell, M. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, **7**, 155–162. (Cited on pages 19, 177, 216 and 291).

Printz, H., & Olsen, P. 2002. Theory and Practice of Acoustic Confusability. *Computer Speech and Language*, **16**(1), 131–164. (Cited on pages 28 and 293).

Rabiner, L., & Huang, B. H. 1993. *Fundamentals of Speech Recognition*. Prentice-Hall. (Cited on page 97).

Rabiner, L. R. 1990. Readings in speech recognition. Morgan Kaufmann Publishers Inc. (Cited on page 95).

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks*. Cambridge University Press. (Cited on page 276).

Rojas, R. 1996. *Neural Networks - A Systematic Introduction*. Springer-Verlag. (Cited on page 276).

Romero, V., Pastor, M., Toselli, A. H., & Vidal, E. 2006. Improving handwritten off-line text slant correction. *Pages 389–394 of: Proceedings of the International Conference on Visualization, Imaging, and Image Processing*. (Cited on page 132).

Rosenfeld, R. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, **10**, 187–228. (Cited on page 85).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986. Learning internal representations by error propagation. *Pages 319–362 of: PDP: Computational models of cognition and perception, I*. MIT Press. (Cited on pages 37 and 295).

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*, **27**(6), 588–595. (Cited on pages 97, 99 and 305).

Sanchis, G., & Casacuberta, F. 2006. N-Best Reordering in Statistical Machine Translation. *Pages 99–104 of: Proceedings of the Jornadas en Tecnologia del Habla*. (Cited on pages 166, 311 and 313).

Schenk, J., Lenz, J., & Rigoll, G. 2008. On-Line Recognition of Handwritten Whiteboard Notes: A Novel Approach for Script Line Identification And Normalization. *Pages 540–543 of: Proceedings of the International Workshop on Frontiers in Handwriting Recognition*. (Cited on page 133).

Schwenk, H. 2010. Continuous space language models for statistical machine translation. *The Prague Bulletin of Mathematical Linguistics*, **93**. (Cited on pages 5, 67 and 289).

Schwenk, H., Rousseau, A., & Attik, M. 2012. Large, Pruned or Continuous Space Language Models on a GPU for Statistical Machine Translation. *Proceedings of the annual meeting of the Nord American chapter of the Association for Computational Linguistics on Human Language Technologies*. (Cited on page 44).

Schwenk, H. 2007. Continuous space language models. *Computer Speech and Language*, **21**(3), 492–518. (Cited on pages 5, 27, 35, 37, 39, 40, 48, 50, 60, 67, 68, 109, 289, 294, 295, 296, 297, 298 and 300).

Schwenk, H., & Gauvain, J.-L. 2002. Connectionist language modeling for large vocabulary continuous speech recognition. *Pages 765–768 of: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. (Cited on pages 5, 27, 109 and 289).

Schwenk, H., & Gauvain, J.-L. 2005. Training Neural Network Language Models on Very Large Corpora. *In: Proceedings of the Empirical Methods in Natural Language Processing Conference*. (Cited on page 109).

Schwenk, H., & Koehn, P. 2008. Large and diverse language models for statistical machine translation. *Pages 661–666 of: Proceedings of the International Joint Conference on Natural Language Processing*. (Cited on pages 5, 41, 289 and 296).

Schwenk, H., Déchelotte, D., & Gauvain, J.-L. 2006. Continuous space language models for statistical machine translation. *Pages 723–730 of: Proceedings of the International Conference on Computational Linguistics*. (Cited on pages 5 and 289).

Segarra, E., Castro, M. J., Galiano, I., García, F., Gómez, J. A., Griol, D., Hurtado, L. F., Sanchis, E., Torres, F., & Zamora-Martínez, F. 2008. Adquisición de un corpus de diálogos para un dominio de reservas de instalaciones deportivas. *Pages 163–166 of: Proceedings of the Jornadas en Tecnologia del Habla*. (Cited on page 245).

Simard, P. Y., Steinkraus, D., & Agrawala, M. 2005. Ink normalization and beautification. *Pages 1182–1187 of: Proceedings of the International Conference on Document Analysis and Recognition*. (Cited on pages 132 and 133).

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., & Makhoul, J. 2006. A Study of Translation Edit Rate with Targeted Human Annotation. *In: Proceedings of the Association for Machine Translation in the Americas Conference*. (Cited on pages 16, 17 and 291).

Stolcke, A. 2002. SRILM: an extensible language modeling toolkit. *Pages 901–904 of: Proceedings of the International Conference on Spoken Language Processing*. (Cited on pages 73, 120, 140 and 308).

Stolcke, A., König, Y., & Weintraub, M. 1997. Explicit word error minimization in N-best list rescoring. *Pages 163–166 of: Proceedings of the European Conference on Speech Communications and Technology*. (Cited on page 8).

Thimm, G., & Fiesler, E. 1997. High Order and Multilayer Perceptron Initialization. *IEEE Transactions on Neural Networks*, **8**(2), 349–359. (Cited on page 281).

Tillmann, C., & Ney, H. 2003. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics*, **29**(1), 97–133. (Cited on pages 184, 187 and 313).

Tortajada, S., & Castro, M. J. 2006. Diferentes Aproximaciones a la Codificación del Vocabulario en Modelado de Lenguaje Conexionista. *Pages 59–63 of: Proceedings of the Jornadas en Tecnologia del Habla*. (Cited on page 35).

Toselli, A. H., Juan, A., González, J., Salvador, I., Vidal, E., Casacuberta, F., Keysers, D., & H. Ney. 2004. Integrated Handwriting Recognition and Interpretation using Finite-State Models. *International Journal of Pattern Recognition and Artificial Intelligence*, **18**(4), 519–539. (Cited on page 134).

Uchida, S., Taira, E., & Sakoe, H. 2001. Nonuniform slant correction using dynamic programming. *Pages 434–438 of: Proceedings of the International Conference on Document Analysis and Recognition*, vol. 1. (Cited on page 133).

Vidal, E. 1997. Finite-State Speech-to-Speech Translation. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, **1**, 111. (Cited on page 161).

Vidal, E., Thollard, F., Casacuberta, F., de la higuera, C., & Carrasco, R. 2005. Probabilistic finite-state machines - part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**(7), 1013–1025. (Cited on pages 166 and 167).

Vinciarelli, A., & Luettin, J. 2001. A new normalization technique for cursive handwritten words. *Pattern Recognition Letters*, **22**(9), 1043–1050. (Cited on page 132).

Viterbi, A. J. 1967. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Informatation Theory*, **IT-13**, 260–269. (Cited on pages 53, 97 and 108).

Witten, I. H., & Bell, T. C. 1991. The zero-frequency problem: Estimating the probability of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, **37**, 1085–1094. (Cited on pages 32 and 294).

Wong, K. Y., Casey, R. G., & Wahl., F. M. 1982. Document Analysis system. *IBM Journal of Research and Developement*, **26**(6), 647–655. (Cited on page 132).

Xu, W., & Rudnicky, A. 2000. Can Artificial Neural Networks Learn Language Models? *In: Proceedings of the International Conference on Spoken Language Processing*. (Cited on pages 35 and 294).

Zamora-Martínez, F., & Castro-Bleda, M. J. 2010. Traducción automática basada en Ngramas conexionistas. *Proc. of the Sociedad Española para el Procesaminto del Lenguaje Natural.*, **45**(45), 221–228. (Cited on page 242).

Zamora-Martinez, F., & Castro-Bleda, M. J. 2011. CEU-UPV English-Spanish system for WMT11. *Pages 490–495 of: Proceedings of the Workshop on Statistical Machine Translation*. (Cited on pages 210, 232, 239, 242, 285, 316 and 317).

Zamora-Martinez, F., & Castro-Bleda, M. J. 2012. Integrating Neural Network Language Models in the Decoding Stage of a Machine Translation System. *In: Proceedings of the International Conference on Computational Linguistics*. Submitted. (Cited on pages 232, 239, 241, 316 and 317).

Zamora-Martínez, F., & Sanchis, G. 2010. UCH-UPV English–Spanish System for WMT10. *Pages 207–211 of: Proceedings of the Workshop on Statistical Machine Translation and MetricsMATR*. (Cited on pages 210, 232, 239, 242, 285 and 316).

Zamora-Martínez, F., España, S., Gorbe, J., & Castro, M. J. 2006. Entrenamiento de Modelos de Lenguaje Conexionistas con Grandes Vocabularios. *Pages 141–144 of: Proceedings of the Jornadas en Tecnologia del Habla*. (Cited on page 243).

Zamora-Martínez, F., España, S., & Castro, M. J. 2007. Behaviour-based Clustering of Neural Networks applied to Document Enhancement. *Pages 144–151 of: Computational and Ambient Intelligence*. Lecture Notes in Computer Science, vol. 4507. Springer. (Cited on pages 132 and 245).

Zamora-Martínez, F., Castro-Bleda, M. J., Tortajada-Velert, S., & España-Boquera, S. 2009a. A Connectionist approach to Part-Of-Speech Tagging. *Pages 421–426 of: Proceedings of the International Conference on Neural Computation*. (Cited on page 245).

Zamora-Martínez, F., Castro-Bleda, M. J., Tortajada-Velert, S., & España-Boquera, S. 2009b. Adding morphological information to a connectionist Part-Of-Speech tagger. *Pages 169–178 of: Proceedings of the Conferencia de la Asociación Española para la Inteligencia Artificial*. (Cited on page 245).

Zamora-Martínez, F., Castro-Bleda, M. J., & España-Boquera, S. 2009. Fast Evaluation of Connectionist Language Models. *Pages 33–40 of: Proceedings of the International Work-Conference on Artificial Neural Networks*. Lecture Notes in Computer Science, vol. 5517. Springer. (Cited on pages 81, 237, 241, 301, 302 and 303).

Zamora-Martínez, F., Castro-Bleda, M. J., España-Boquera, S., & Gorbe-Moya, J. 2010. Improving isolated handwritten word recognition using a specialized classifier for short words. *In: Current Topics in Artificial Intelligence*. Lecture Notes in Artificial Intelligence. Springer-Verlag. (Cited on page 244).

Zamora-Martinez, F., Castro-Bleda, M. J., & Schwenk, H. 2010. N-gram-based Machine Translation enhanced with Neural Networks for the French-English BTEC-IWSLT'10 task. *Pages 45–52 of: Proceedings of the International Workshop on Spoken Language Translation*. (Cited on pages 219, 232, 239, 242, 285, 316 and 317).

Zamora-Martínez, F., Castro-Bleda, M. J., España Boquera, S., & Gorbe-Moya, J. 2010. Unconstrained Offline Handwriting Recognition using Connectionist Character N-grams. *Pages 4136–4142 of: Proceedings of the International Joint Conference on Neural Networks*. (Cited on pages 128, 155, 238, 241, 308, 309 and 310).

Zamora-Martínez, F., España-Boquera, S., Castro-Bleda, M. J., & de Mori, R. 2012a. Cache Neural Network Language Models based on Long-Distance Dependencies for a Spoken Dialog System. *Pages 4993–4996 of: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. (Cited on pages 89, 125, 237, 238, 242, 304, 306 and 307).

Zamora-Martínez, F., Castro-Bleda, M. J., & España-Boquera, S. 2012b. Integrating efficiently Neural Network Language Models on Viterbi based decoders. *IEEE Transactions on Neural Networks*. Forthcoming. (Cited on pages 81, 241 and 309).

Zamora-Martínez, F., Frinken, V., España-Boquera, S., Castro-Bleda, M. J., Fisher, A., & Bunke, H. 2012c. Neural Network Language Models for Off-Line Handwriting Recognition. *Pattern Recognition*. Submitted. (Cited on pages 128, 155, 238, 241, 308 and 310).

Zens, R., Ney, H., Watanabe, T., & Sumita, E. 2004. Reordering constraints for phrase-based statistical machine translation. *Page 205 of: Proceedings of the International Conference on Computational Linguistics*. (Cited on page 187).

Zipf, G. K. 1949. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley. (Cited on page 50).

Apéndices

A

GLOSSARY OF SYMBOLS AND ACRONYMS

Contents

A.1 Mathematical symbols

$\bar{v}$	v is a sequence of items (or a vector).
$\mathbf{V}$	V is a matrix.
$\|\cdot\|$	Sequence length, or set cardinality.
$S(\{\bar{v}^{(1)}, \ldots, \bar{v}^{(K)}\}) = \sum_{i=1}^{K} \|\bar{v}^{(i)}\|$	Sum of sequence lengths.
$\bar{x} = x_1 x_2 \ldots x_{\|\bar{x}\|}$	System input, feature vector sequence.
x_i	i-th feature vector.
$\bar{y} = y_1 y_2 \ldots y_{\|\bar{y}\|}$	System hypothesis, that is, a word sequence.
y_i	i-th word from a system hypothesis.
$\mathcal{Y} = \{\bar{y}^{(1)}, \ldots, \bar{y}^{(K)}\}$	System output for a set of K inputs.
$\bar{y}^{(i)} = y_1^{(i)} y_2^{(i)} \ldots y_{\|\bar{y}^{(i)}\|}^{(i)}$	System output for i-th input.
$y_j^{(i)}$	j-th word from i-th system output.
$\mathcal{R} = \{\bar{r}^{(1)}, \bar{r}^{(1)}, \ldots, \bar{r}^{(K)}\}$	Set of K reference sentences.
$\bar{r}^{(i)} = r_1^{(i)} r_2^{(i)} \ldots r_{\|\bar{r}^{(i)}\|}^{(i)}$	Reference for i-th sentence.
$r_j^{(i)}$	j-th word from i-th reference sentence.
$p(\bar{y}\|\bar{x})$	Posterior probability.
$p(\bar{x}\|\bar{y})$	Likelihood.
$p(\bar{y})$	Prior probability.
$\mathcal{C}(\cdot)$	Counting function.
Ω	Task vocabulary.
Ω'	Short-List vocabulary for NN LMs.
Ω^I	NN LM input vocabulary.
ω	Word from Ω.
$\bar{L}_j$	Local encoding of an input word.
$\bar{I}$	Input layer of the NN LM, $\bar{I} = \bigcup_j \bar{L}_j$.
$\bar{P}_j$	Projection of input $\bar{L}_j$.
$\bar{H}$	NN LM hidden layer.
h_i	Hidden layer neron i.
$\bar{O}$	NN LM output layer.
o_i	Outpu layer neuron i.
$\bar{T}$	NN LM target output.
t_i	Target output for neuron i.
$\mathbf{W}_{L,P}$	Weight matrix from projection layer.
$\bar{B}_P$	Bias vector from projection layer.
$\mathbf{W}_{P,H}$	Weight matrix from hidden layer.
$\bar{B}_H$	Bias vector from hidden layer.
$\mathbf{W}_{H,O}$	Weight matrix from output layer.
$\bar{B}_O$	Bias vector from output layer.
N	N-gram order.
$\bar{h}_k$	Context for N-gram k.
$\bar{h}_1^{u-1}$	NN LM cache module.
$\mathcal{H}_m(\cdot)$	Log-linear combination model.
$\bar{\mathcal{T}} = \mathcal{T}_1 \mathcal{T}_2 \ldots \mathcal{T}_{\|\bar{\mathcal{T}}\|}$	Sequence of bilingual tuples.

$\varphi(\cdot)$ Mapping function between input source sentence positions and tuple indices.

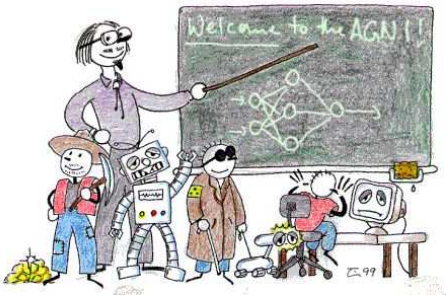
A.2 Acronyms

ANN	Artificial Neural Network.
ASR	Automatic Speech Recognition.
bcc	Begin Context Cue.
BLEU	Bilingual Language Evaluation Understudy.
BP	Back-propagation algorithm.
CER	Character Error Rate o Concept Error Rate.
Cache NN LM	Cache Neural Network Language Model.
CRF	Conditional Random Field.
FFT	Fast Fourier Transform.
FIR	Finite Impulse Response.
F_s	Sampling frequency.
ecc	End Context Cue.
GIATI	Grammar Inference and Alignments for Transducer Inference.
GLI	Generalized Linear Interpolation.
GLI-CS	Generalized Linear Interpolation on Continuous Space.
GPU	Graphical Processor Unit.
GSF	Grammar Scale Factor.
HMM	Hidden Markov Model.
HMM/ANN	Hybrid Hidden Markov Model with Artificial Neural Networks.
HTR	Handwritten Text Recognition.
LBLM	Log-Bilinear Laguage Model.
HLBLM	Hierarchical Log-Bilinear Laguage Model.
IWSLT	International Workshop on Spoken Language Translation.
LI	Linear Interpolation.
LM	Language Model.
LMkey	State identifier of an N-grama LM.
LSTM	Long-Short Term Memory.
MAP	Maximum A Posteriori
MBR	Maximum Bayes Risk
MERT	Minimum Error Rate Training.
MSE	Mean Square Error.
MLP	Multilayer Perceptron.
NN LM	Neural Network Language Model.
RBMs	Restricted Boltzmann Machines.
ROVER	Recognizer Output Voting Error Reduction.
SER	Sentence Error Rate.
SFST	Stochastic Finite State Transducer.
SLU	Spoken Language Understanding.
SMT	Statistical Machine Translation.
SOUL NN LM	Structured OUtput Layer Neural Network Language Model.
TER	Translation Edit Error.

TIP	Tuple Insertion Penalty.
TrieLM	Data structure to represent a NN LM as a finite state automaton.
TSP	Travelling Salesman Problem.
WER	Word Error Rate.
WIP	Word Insertion Penalty.
WMT	Worshop of Machine Translation.

B

REDES NEURONALES ARTIFICIALES



Índice

Este apéndice introduce los conceptos básicos de las redes neuronales artificiales, para después exponer el algoritmo de entrenamiento back-propagation. La notación en este apéndice es autocontenida, y por tanto no deben confundirse con la utilizada en el resto de la tesis.

B.1 El problema del aprendizaje

Vamos a ver la red neuronal como una caja negra (figura B.1), con n entradas y m de salidas. La red puede contener tantas unidades ocultas como se quiera, y puede exhibir cualquier patrón de conexiones entre las unidades, que en este momento vamos a obviar. Definimos un conjunto de patrones (dataset) como $\{(x_1, t_1), \ldots, (x_p, t_p)\}$, que consiste en p pares de vectores, uno de dimensión n y otro de m, que son denominados el patrón de entrada y el patrón de salida. Cuando llega un patrón de entrada x_i, se produce una salida o_i. Lo que queremos es que o_i aproxime a t_i para todo $i = 1, \ldots, p$ utilizando un algoritmo de aprendizaje. Esto es, queremos minimizar una función de error de la red que mide cuanto nos aproximamos a la función objetivo.

Figura B.1: *Red neuronal como una caja negra.*

Existen diversas funciones de error para medir lo buena que es la aproximación de la red a la función objetivo. Las más importantes son:

Error cuadrático medio (MSE):

$$E = \frac{1}{2p}\sum_{i=1}^{p}\sum_{j=1}^{m} ||t_{i,j} - o_{i,j}||^2$$

Entropía Cruzada (Cross-Entropy):

$$E = \frac{1}{p}\sum_{i=1}^{p}\sum_{j=1}^{m} t_{i,j} \log o_{i,j}$$

Verosimilitud condicional:

$$E = \frac{1}{2p}\sum_{i=1}^{p}\sum_{j=1}^{m} \left(t_{i,j} \log \frac{t_{i,j}}{o_{i,j}(x_i)} + (1 - t_{i,j}) \log \frac{1 - t_{i,j}}{1 - o_{i,j}(x_i)} \right)$$

Error cuadrático normalizado:

$$E = \frac{\sum_{i=1}^{p}\sum_{j=1}^{m} ||t_{i,j} - o_{i,j}||^2}{\sum_{i=1}^{p}\sum_{j=1}^{m} (t_{i,j} - t_j^*)} \qquad t_j^* = \frac{\sum_{i=1}^{p} t_{i,j}}{p}$$

Error de Minkowski:

$$E = \sum_{i=1}^{p} \sum_{j=1}^{m} ||t_{i,j} - o_{i,j}||^R$$

El MSE y la entropía cruzada son los más comunes dado que presentan algunas características convenientes, como por ejemplo permitir que la red estime probabilidades.

B.2 Redes neuronales

Una neurona es una unidad de proceso dotada de una serie de entradas y de una única salida. La neurona produce una combinación lineal de las entradas, y activa su salida con un valor que depende de esta combinación de un modo en general no lineal mediante una función de activación. Esta unidad se denomina perceptrón, y realiza las siguientes operaciones:

- Un producto escalar del vector de entradas, $\bar{x}$, por otro vector que denominaremos de pesos, $\bar{w}$. Además le resta un valor que denominaremos umbral θ. Al resultado se le denomina potencial: $\gamma = \bar{x} \cdot \bar{w} - \theta$.
- Al resultado anterior le aplica una función de activación f, que en general interesa que no sea lineal: $f(\gamma) = z$.
- El resultado z será propagado por la salida de la neurona.

Para simplificar los cálculos, vamos a añadir a la neurona una nueva entrada, x_0, que estará conectada a un 1 fijo, de manera que el peso w_0 hará la función del umbral θ:

$$f(\sum_{i=0}^{n} w_i x_i) = z$$

Podemos encontrar las siguientes funciones de activación implementadas en `April`:

- **Lineal**: es la función identidad. Por tanto: $f(x) = x$.
- **Sigmoide**: es la más clásica de todas. Proporciona valores entre 0 y 1, y tiene un gran parecido con la función escalón. El valor del factor c indica cuánto se parece al escalón de manera que cuanto mayor es el valor de c mayor es el parecido (véase figura B.3). El cálculo es el siguiente: $s_c(x) = \dfrac{1}{1 + exp(-cx)}$.
- **Tangente hiperbólica**: es una función con la misma forma que la sigmoide, pero toma valores entre $[-1, 1]$. En realidad entre ambas funciones se puede encontrar una transformación lineal que convierta la una en la otra. Queda de esta forma: $f(x) = tanh(x) = \dfrac{exp(x) - exp(-x)}{exp(x) + exp(-x)}$.
  Para $c = 1$, $tanh(x) = 2s_1(2x) - 1$.
- **Softmax**: esta función normaliza las salidas de las neuronas para que la suma de todas ellas sea igual a 1. Dado un conjunto de m salidas en la red, cada salida k se calcula así: $f(x_k) = exp(x_k) / \sum_{k'=1}^{m} exp(x_{k'})$.

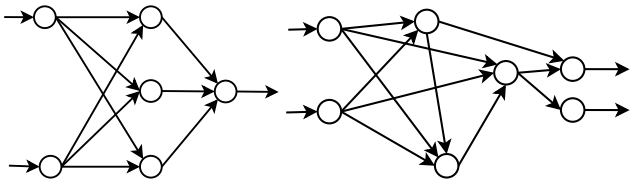


Figura B.2: *Diagrama de redes hacia-delante (una capa a capa y otra general).*

Un conjunto de neuronas conectadas entre sí es lo que se conoce como Red Neuronal Artificial o "Artificial Neural Network" (ANN). La topología clásica agrupa las neuronas por capas, conectando únicamente todas las neuronas de una capa con todas las de la siguiente. A este tipo de red se la denomina capa a capa, pero puede ser generalizada, conectando neuronas que estén en capas no contiguas. De esta forma, se convierte la red en un grafo acíclico, pasando a denominarse red hacia-delante general (véase figura B.2).

Una red neuronal puede ser ajustada modificando los valores de los pesos de sus conexiones. Para realizar este ajuste existen diversas técnicas. Las más utilizadas son técnicas basadas en el descenso por gradiente. Por lo tanto, es necesario calcular el gradiente de la función de error (derivada), siendo una condición indispensable que la función de activación de las neuronas sea derivable.

B.3 Algoritmos de aprendizaje

Existen dos tipos de algoritmos para ajustar los pesos de las redes. El primero es el de los métodos basados en descenso por gradiente, y el segundo el de los métodos de segundo orden, que suelen ser variaciones de los métodos del primer tipo. Vamos a hacer una pequeña revisión de diferentes algoritmos para ambos métodos.

Métodos basados en el descenso por gradiente

El algoritmo más conocido es el Error Back-propagation (BP) y todas sus variantes. Estos métodos consisten en definir una función de error y calcular su gradiente respecto a los parámetros o pesos de la red. De esa forma, los pesos son modificados en la dirección del gradiente, buscando un mínimo en la función de error. Existen diversas variantes, entre las más importantes:

- BP batch: es el algoritmo clásico. El error de la red es la media de los errores producidos por cada muestra de entrenamiento. La actualización de los pesos se realiza cuando ya se han visitado todas las muestras.
- BP on-line: sólo difiere del anterior en el momento en que deben ser actualizados los pesos. En este caso se actualizan con cada muestra de entrenamiento, calculando el error únicamente para esa muestra.

Métodos de segundo orden

En esta sección vamos a ver variantes del BP que tratan de acelerar la velocidad de convergencia.

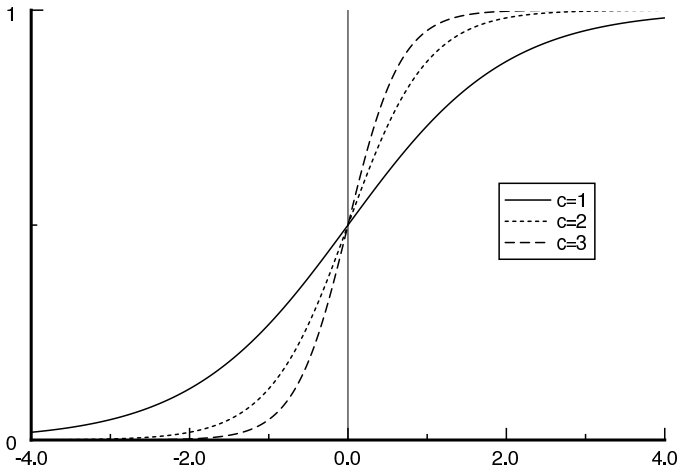


Figura B.3: *Tres sigmoides (con c=1, c=2 y c=3).*

- Algoritmo *Quickprop*: sigue los mismos pasos que el BP, pero para calcular el incremento de los pesos ajusta una función cuadrática mediante la derivada de la función de error en la iteración anterior y la actual. De esa forma, modifica los pesos buscando el mínimo de la función cuadrática, lo que produce un descenso mucho más rápido hacia el mínimo de la función de error.

- Algoritmo de *Newton*: para estudiar este algoritmo necesitamos de un nuevo operador, la *matriz de Hessian*:

$$H = \frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}}.$$

  Esta matriz contiene la derivada segunda de la función de error respecto a cada uno de los pesos. A partir de la inversa de esta matriz se puede calcular cual es la dirección exacta del mínimo de la función de error E. El problema de este método es que el cálculo exacto de la inversa de H tiene un coste computacional de $O(W^3)$, siendo W el número de pesos en la red.

- Algoritmos *quasi-Newton*: estos métodos aceleran la velocidad del método anterior aproximando el cálculo de la inversa para que no sea tan costoso, introduciendo así un factor de imprecisión en los cálculos. Reduce el coste computacional a $O(W^2)$. El resto del método es exactamente el mismo.

B.4 Introducción al algoritmo BP

El BP es un algoritmo de aprendizaje supervisado que busca la mejor solución a un problema utilizando la técnica del descenso por gradiente. El texto que se va a presentar ha sido extraído de los libros [Bishop, 1995; Ripley, 1996; Rojas, 1996].

Necesitamos que las funciones de activación sean derivables, siendo la más popular la sigmoide, definida como sigue: $s_c : \Re \rightarrow [0, 1]$

$$s_c(x) = \frac{1}{1 + exp(-cx)}$$

La constante c será seleccionada arbitrariamente, aunque siempre con valores mayores que cero, y sirve para modificar la forma de la función, de manera que cuando $c \to \infty$ la función sigmoide converge hacia la función escalón en el origen de coordenadas (véase figura B.3). Para simplificar las expresiones, vamos a asumir que $c = 1$, de manera que a partir de ahora $s_1(x)$ será $s(x)$. Esta simplificación se puede realizar sin perder generalidad en el resultado.

La derivada de la función sigmoide respecto a x es la siguiente:

$$\frac{d}{dx}s(x) = \frac{exp(-x)}{(1 + exp(-x))^2} = s(x)(1 - s(x))$$

B.4.1. El problema del aprendizaje

Las redes hacia-delante se pueden ver como un grafo computacional cuyos nodos son unidades de proceso y cuyas aristas son dirigidas y transmiten información numérica de un nodo a otro. Cada unidad es capaz de calcular una función en general no lineal de sus entradas, de manera que la red computa una composición de funciones que transforma el espacio de entrada en el espacio de salida. A esta función se le denomina función de la red. El problema del aprendizaje consiste en encontrar una combinación de pesos de manera que la función de red se aproxime lo más posible a una cierta función f. Sin embargo, no conocemos la función f, sino sólo ejemplos que la definen implícitamente. Por tanto, enlazando con el primer apartado del capítulo, necesitamos ajustar los parámetros de la red (pesos) parar minimizar una función de error determinada (en nuestro caso MSE) que es la que nos da una medida de lo buena que es la aproximación que estamos haciendo.

Tras minimizar la función para el conjunto de entrenamiento, cuando llegue un patrón de entrada desconocido hasta el momento, se espera que la red interpole y acierte con la salida. La red aprende a reconocer similitudes entre el vector de entrada y los patrones que ya ha aprendido, de manera que trata de obtener una salida similar.

Vamos a extender la red para que cada vez que le llega un nuevo patrón, calcule la función de error E para el mismo. Para ello, conectaremos cada unidad de salida a un nuevo nodo que evalúa la función $\frac{1}{2}(o_{ij} - t_{ij})^2$, donde o_{ij} y t_{ij} denotan la componente j del vector de salidas o_i y del objetivo t_i. La salida de estas m unidades es recogida por un nodo final que suma todas las entradas que recibe, de manera que la salida de este nodo es la función E_i. Esta extensión se realiza para cada patrón t_i. La salida de la red extendida de esta forma es la función de error E para cada muestra.

Ahora tenemos una red capaz de calcular el error para un conjunto de entrenamiento. Los pesos de la red son el único parámetro que se puede modificar para minimizar la función E. Dado que E se calcula a partir de la composición de las unidades de la red, será una función continua y derivable de los l pesos $w_1, w_2, \ldots, w_l$ que la componen. Por tanto, se puede minimizar E mediante un proceso iterativo de descenso por gradiente. El gradiente se define así:

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_l}).$$

Cada peso es modificado por el incremento:

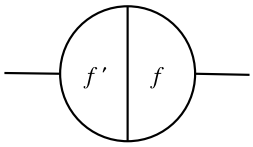


Figura B.4: *Los dos lados de una unidad de proceso.*

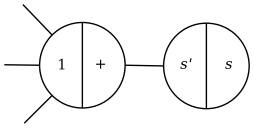


Figura B.5: *Sumatorio y función de activación separadas en dos unidades.*

$$\triangle w_i = -\eta \frac{\partial E}{\partial w_i}, \text{ para } i = 1, .., l.$$

Donde η representa el factor de aprendizaje, un parámetro de proporcionalidad que permite definir la longitud del paso de cada iteración en la dirección del gradiente.

Con esta extensión de la red original, el problema del aprendizaje se reduce a un mero cálculo del gradiente de la función de red con respecto a los pesos. Por tanto, el objetivo es encontrar un punto donde $\nabla E = 0$, que coincidirá con uno de los mínimos locales de la función de error E.

B.4.2. Derivada de la función de red

Vamos a dejar de lado el problema del aprendizaje, y vamos a centrarnos en encontrar la derivada de la función de red extendida respecto a cada uno de los pesos, es decir, como obtener ∇E.

La red realiza dos pasos, el paso hacia-delante y el paso de retropropagación. Las unidades, cuando trabajan hacia-delante calculan una función de sus entradas y propagan el resultado en la salida. Al mismo tiempo, calculan la derivada de la misma función para la misma entrada, y guardan el resultado para su posterior uso. Podríamos ver las unidades como separadas en dos partes, una con el resultado de la función y otra con la derivada, como se puede ver en la figura B.4. En el paso de retropropagación, las unidades utilizarán este último resultado para calcular la derivada de la función de red.

Separaremos ahora la unidad en dos independientes (figura B.5), la primera que calcula, para una unidad k, el sumatorio de todas sus entradas, obteniendo: $\gamma_k = \sum_i w_{ki} \cdot x_{ki}$. La segunda calculará la función primitiva de la misma unidad (por ejemplo, la sigmoide), procesando por tanto: $f(\gamma_k) = z_k$.

De esta forma, tendremos tres casos en los que estudiar como calcular la derivada:

Primer caso: composición de funciones. La composición de funciones se obtiene mediante la unión de la salida de una unidad a la entrada de otra. Denominaremos a la función del primer nodo g, y a la del segundo f. Entonces, la red computa la composición $f(g(x))$. El resultado de la composición es obtenido por la salida de la última unidad. Al mismo tiempo,

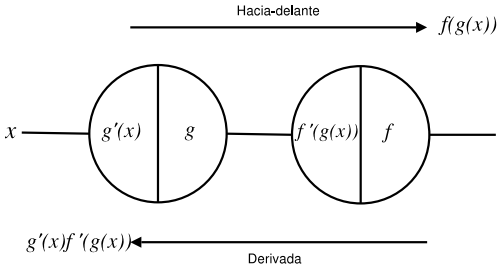


Figura B.6: *Composición de funciones.*

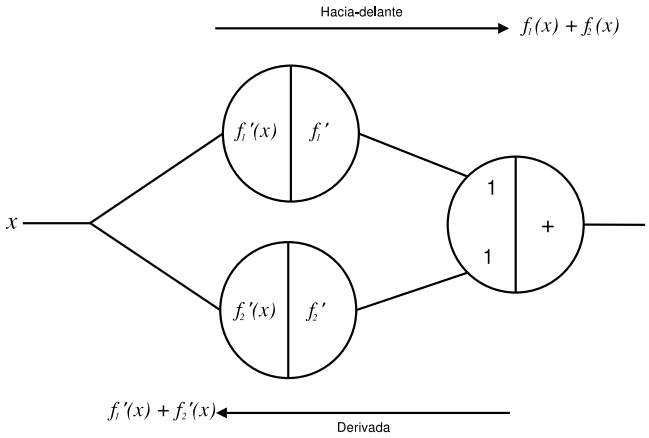


Figura B.7: *Adición de funciones.*

cada unidad guarda el valor de la derivada en su lado izquierdo. En el paso de retropropagación entra un 1 por el lado derecho de la red. Este valor es multiplicado por el valor guardado en el lado izquierdo de cada unidad. El resultado es transmitido hacía la siguiente unidad a la izquierda. Por lo tanto, en el caso de ejemplo, la red estará calculando $f'(g(x))g'(x)$, que es la derivada de $f(g(x))$. Cualquier secuencia de composición de funciones puede ser evaluada siguiendo este esquema (figura B.6).

Segundo caso: adición de funciones. En este caso, tendremos un nodo salida que realiza la suma de todas sus entradas, y una serie de nodos conectados a él en paralelo. Veremos el caso para dos nodos de entrada, que calculan las funciones f_1 y f_2. El nodo de salida, calculará por lo tanto $f_1(x) + f_2(x)$. La derivada parcial de la función de adición respecto a cualquiera de sus entradas es 1. Por lo tanto, cuando se ejecuta el paso de retropropagación el resultado final de la red va a ser $f_1'(x) + f_2'(x)$, que es la derivada de la función de adición anterior. Por inducción se puede demostrar que la derivada de la adición de cualquier número de funciones puede ser tratada siguiendo estos pasos (figura B.7).

Tercer caso: pesos en las aristas. Las aristas pueden tener un peso asociado, que podría ser tratado como un caso de composición de funciones, pero es más fácil tratar con ellas de forma diferente. En el paso hacia-delante la información x es multiplicada por el peso w. El

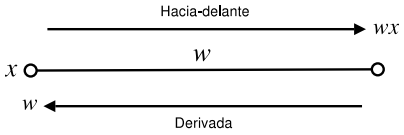


Figura B.8: *Pesos en las aristas.*

resultado es wx. En el paso de retropropagación el valor 1 es multiplicado por el peso de la arista. El resultado es w, que es la derivada de wx respecto a x (figura B.8).

B.4.3. Algoritmo de BP

Ya podemos formular completamente el algoritmo BP. Vamos a asumir que estamos tratando con una red que únicamente tiene una unidad de entrada y otra de salida.

Consideraremos una red con una única entrada real x y una función de red F. La derivada $F'(x)$ es calculada en dos fases:

- Hacia-delante: la entrada x se introduce en las entradas de la red, siguiendo un orden topológico. Cada nodo calcula su función primitiva y la derivada. Guardamos la derivada para después.
- retropropagación: la constante 1 es introducida por el lado derecho de la red, por la unidad de salida. La información pasa de un nodo a otro y es multiplicada por el valor del lado izquierdo de cada unidad. El resultado recogido por la unidad de entrada es la derivada de la función de red respecto a x.

B.4.4. Aprendiendo con el BP

Vamos a considerar de nuevo el problema del aprendizaje en las redes neuronales. Queremos minimizar una función E que depende de los pesos de la red. El paso hacia-delante es computado tal y como se ha explicado, pero vamos a guardar, además del valor de la derivada en el lado izquierdo de cada unidad, el valor de la salida en el lado derecho. El paso de retropropagación será ejecutado por la red extendida que calcula la función de error, y fijaremos nuestra atención en uno de los pesos, que denominaremos w_{ij}, asociado a la arista que comienza en la unidad i y llega a la j. Por tanto, en el primer paso calcularemos $x_j = \sum o_i w_{ij}$, de manera que x_j será el valor que llegará a la unidad j. Asimismo tendremos que $o_j = f_j(x_j)$, siendo f_j su función de activación. La unidad i guardará en su salida el valor o_i. Veamos ahora como calcular las derivadas parciales de E:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ij}} = o_i \frac{\partial E}{\partial x_j} = o_i f_j'(x_j)\frac{\partial E}{\partial o_j} = o_i \delta_j, \text{ con } \delta_j = f_j'(x_j)\frac{\partial E}{\partial o_j}.$$

La primera igualdad es fruto de la dependencia de E en los pesos sólo a través de las salidas. La segunda se obtiene a partir de $x_j = \sum o_i w_{ij}$. Denominaremos a δ_j el *error retropropagado* para el nodo j. Por lo tanto cada nodo necesitará guardar la siguiente información para poder realizar estos cálculos:

- La salida o_i del nodo en el paso hacia-delante.

- El resultado acumulativo tras computar el paso de retropropagación en ese nodo, δ_i.

Para concluir el cálculo es necesario definir cual va a ser el valor δ_j para cada unidad de la red. Podemos observar que $\delta_j = f'_j(x_j)\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial x_j}$. De esta expresión, $f'_j(x_j)$ la podemos calcular para cualquier unidad, ya que en el paso de retropropagación conocemos el valor de x_j para todas las unidades. Despejaremos cual debe ser el valor del otro factor de la expresión, distinguiendo dos casos. En las unidades de salida de la red, $\frac{\partial E}{\partial o_j}$ se calcula directamente. Para las unidades de las capas ocultas, teniendo en cuenta la unidad j y las K que le suceden ($j \rightarrow K$):

$$\frac{\partial E}{\partial o_j} = \sum_{k:j\rightarrow K} w_{jk}\frac{\partial E}{\partial x_k} = \sum_{k:j\rightarrow K} w_{jk}\delta_k.$$

Una vez han sido computadas todas las derivadas parciales, podemos añadir a cada peso w_{ij} el incremento:

$$\triangle w_{ij} = -\eta o_i \delta_j.$$

Con todo esto ya tenemos definido el funcionamiento del algoritmo para redes generales, capaces de describir cualquier topología hacia-delante.

B.5 Coste del entrenamiento

El entrenamiento consta de dos pasos, el paso hacia-delante y el paso de retropropagación. El primero de ellos tiene un coste de $\Theta(W)$, siendo W el número de conexiones en la red. El paso de retropropagación calcula primero las derivadas, con un coste de $\Theta(N)$, siendo N el número de neuronas en la red. Después modifica los pesos aplicando el ajuste necesario, con un coste de $\Theta(W)$. Por lo tanto, el coste final del algoritmo será de $\Theta(2(W+N))$, y dado que normalmente $N << W$, el coste es lineal con el número de conexiones en la red, $\Theta(W)$.

El coste no se puede reducir dado que para modificar la red es necesario visitar todos los pesos. Pero se puede acelerar la ejecución del algoritmo mediante implementaciones pensadas para funcionar eficientemente en un ordenador suponiendo una arquitectura de un tipo más o menos determinado. En este trabajo hemos supuesto una arquitectura con jerarquía de memoria (cache, etc.) serie pero con una CPU capaz de mejorar el rendimiento de las operaciones en coma flotante mediante segmentación del ciclo de instrucción, replicación de operadores, etc. Al mismo tiempo, se han realizado mejoras a nivel de uso de instrucciones de tipo vectorial a través de interfaces para operaciones con matrices y vectores como BLAS [Lawson *et al.*, 1979], y las bibliotecas ATLAS o MKL.

B.6 Inicialización de los pesos

A partir de las investigaciones de algunos autores [Thimm & Fiesler, 1997] el mejor valor para inicializar los pesos de una red se encuentra en el rango $[-0.77, 0.77]$. En general, usaremos valores de inicialización en este rango o en uno más pequeño.

En el entrenamiento de una red neuronal, el valor inicial de los pesos puede modificar el resultado final de la red, pudiendo hacer que caiga en mínimos locales distintos. Por ello resulta muy interesante poder reproducir exactamente los entrenamientos. Consecuentemente, tener un generador de números pseudo-aleatorios que pueda recuperar su estado se convierte en un punto a favor de la herramienta. El generador que se ha utilizado es el `Mersenne Twister` [Matsumoto & Nishimura, 1998]. Con él se pueden repetir los experimentos de forma exacta, pues asegura la recuperación de la misma secuencia de números aleatorios.

B.7 Sobreentrenamiento de las redes

Uno de los problemas más grandes que presentan las técnicas de reconocimiento de formas, y por tanto, el entrenamiento de redes neuronales, es lo que se denomina *over-fitting* o sobreentrenamiento. Se produce cuando la red aprende de forma muy ajustada un determinado conjunto de muestras. Entonces no consigue generalizar las clases y aumenta el error al clasificar muestras que no han sido utilizadas durante el aprendizaje.

Por esta razón se separan las muestras en varios conjuntos. Los de entrenamiento, que servirán para que la red aprenda, los de validación, que servirán para justar los distintos parámetros de los algoritmos de entrenamiento, y el conjunto de test que servirá para obtener el error final de la red con un conjunto completamente distinto a los utilizados durante el entrenamiento. El sobreentrenamiento se evita parando el proceso de aprendizaje cuando aumenta el error para el conjunto de validación. En la mayoría de los casos, si los conjuntos son homogéneos y las muestras usadas para entrenamiento y validación tienen la suficiente variabilidad, la red será capaz de clasificar correctamente el conjunto de test cuando los resultados con el conjunto de validación sean correctos.

C

PARTICIPACIÓN EN CAMPAÑAS INTERNACIONALES DE EVALUACIÓN DE TRADUCCIÓN AUTOMÁTICA

Índice

Este apéndice recoge los resultados oficiales de las campañas de evaluación de traducción automática en los que se ha participado durante la elaboración de esta tesis. Estas campañas son muy importantes debido a que permiten a los grupos de investigación comparar de forma justa los resultados obtenidos con sus sistemas de traducción. En este apéndice se muestran los resultados oficiales de cada una de las campañas en las que se ha participado: WMT'10, IWSLT'10 y WMT'11.

Tabla C.1: *Resultados oficiales del WMT'10 para la tarea Inglés-Español ordenados por BLEU. En negrita se ha remarcado el sistema presentado en esta tesis. //* WMT'10 official ranking for the English-Spanish translation direction.

Position	System	BLEU lowercase	TER lowercase
1	DCU	30.4	58.7
2	UEDIN	29.6	59.1
3	CAMBRIDGE	29.1	60.7
4	UPV-PRHLT	28.7	60.4
5	JHU	28.1	59.6
6	**UCH-UPV**	**28.0**	**61.0**
7	SFU	24.3	62.0
8	DFKI	23.7	65.0
9	KU	21.4	67.5
10	UK-DAN	21.1	68.7

C.1 Participación en el WMT'10

El "Workshop of Machine Translation" del 2010 se celebró en Uppsala (Suecia). Se participó con el sistema descrito en la sección 11.2.2. El sistema presentado quedó en sexta posición, ordenando por BLEU en minúsculas (tabla C.1). La descripción del sistema se publicó en [Zamora-Martínez & Sanchis, 2010].

C.2 Participación en el IWSLT'10

El "International Workshop on Spoken Language Translation" del 2010 se celebró en París (Francia). Se participó con el sistema descrito en la sección 11.3.1. El sistema presentado quedó en segunda posición, según el criterio de la organización, consistente en una combinación de diferentes médidas de error (tabla C.2). La descripción del sistema fue publicada en [Zamora-Martinez *et al.*, 2010].

C.3 Participación en el WMT'11

El "Workshop of Machine Translation" del 2011 se celebró en Edimburgo (Escocia). Se participó con el sistema descrito en la sección 11.2.3. El sistema presentado quedó en segunda posición, ordenando por BLEU en minúsculas (tabla C.3). La descripción del sistema fue publicada en [Zamora-Martinez & Castro-Bleda, 2011].

Tabla C.2: *Resultados oficiales del IWSLT'10 para la tarea BTEC Francés-Inglés ordenados siguiendo el criterio de la organización. En negrita se marca el sistema presentado en esta tesis. //* IWSLT'10 official ranking for the French-English BTEC task.

test09

Position	System	BLEU case+punc	TER
1	MIT	63.6	21.9
2	**DSIC-UPV**	**63.6**	**22.1**
3	UPC	62.0	22.9
4	QMUL	61.8	23.5
5	NICT	61.6	23.0
6	TOTTORI	58.9	24.8
7	KIT	59.9	25.0
8	INESC-ID	59.5	24.0
9	UVA-ISCA	38.6	36.7

test10

Position	System	BLEU case+punc	TER
1	MIT	55.7	25.5
2	**DSIC-UPV**	**53.6**	**25.8**
3	NICT	53.6	26.2
4	UPC	53.3	26.3
5	KIT	51.6	28.1
6	INESC-ID	52.3	27.3
7	QMUL	53.6	27.0
8	TOTTORI	52.0	28.2
9	UVA-ISCA	32.3	39.8

Tabla C.3: *Resultados oficiales del WMT'11 para la tarea Inglés-Español ordenados por BLEU. En negrita se ha remarcado el sistema presentado en esta tesis. //* WMT'11 official ranking for the English-Spanish translation direction.

Position	System	BLEU lowercase	TER lowercase
1	UEDIN	31.9	57.8
2	**UCH-UPV**	**31.5**	**57.9**
3	UU	31.0	58.3
4	UOW	30.3	58.4
5	PROMT	29.6	59.2
6	TRANSDUCTIVEMOSES	28.4	60.8
7	UK-DAN	27.6	62.1
8	GTH-UPM	21.7	69.2

D

ENGLISH SUMMARY

Contents

This appendix is a full summary of this thesis. It is written in English to help international readers to understand this work contributions. Each English section is a chapter in the original text. Some equations are rewritten here, but tables and figures are not, they are referenced to its page at the original text. I apologize for possible troubles due to this "special" conditions, and I hope that intenational readers could understand clearly this work with this summary and the published papers.

D.1 Introduction

Statistical pattern recognition is built over two information sources:

- the likelihood of an input pattern given an output hypothesis,
- and the a priori probability of that hypothesis, computed using Language Models (LMs).

Currently, the scientific community is paying attention to Neural Network Language Models (NN LMs) [Schwenk & Gauvain, 2002; Bengio *et al.*, 2003; Castro-Bleda & Prat, 2003; Schwenk *et al.*, 2006; Schwenk, 2007; Bengio, 2008; Schwenk & Koehn, 2008; Schwenk, 2010]. NN LMs are a kind of continuous space LM, which computes the LM probabilities projecting the input linguistic units on a multidimensional continuous metric space.

Along years, the community did not pay attention to NN LMs due to their computational problems. Nevertheless, the new vectorial CPUs, with more than one core, allows a very important training cost reduction. Though, computational cost problems remain on the evaluation stage of NN LMs, and their coupled integration into a decoder is a big problem. The most extended approach to include NN LMs in a pattern recognition system is via a decoupled N-best rescoring step [Morin & Bengio, 2005; Schwenk, 2007; Emami & Mangu, 2007; Park *et al.*, 2010].

This thesis is focused in the decoding integration problem, developing a novel idea based on the precomputation of softmax normalization constants (see section D.4). New extensions to the standard NN LMs are also studied. Finally, two recognition systems were developed, one for sequence recognition problems, and another for statistical machine translation problems, applied to three different tasks:

- Automatic Speech Recognition (ASR) and Spoken Language Understanding (SLU),
- Handwriting Text Recognition (HTR),
- and Phrase-based and N-gram-based Statistical Machine Translation (SMT).

D.1.1. Pattern Recognition Basis

The goal of a pattern recognition system is to obtain the transcription as a sequence of words from a vocabulary Ω, given a sequence of input feature vectors. The input is denoted as $\bar{x} = x_1 x_2 \dots x_{|\bar{x}|}$ of length $|\bar{x}|$. The system hypothesis is denoted by a word sequence $\bar{y} = y_1 y_2 \dots y_{|\bar{y}|}$ of length $|\bar{y}|$.

In this thesis, for sequence recognition tasks, the input $\bar{x}$ is formed by preprocessed data extracted from speech (Automatic Speech Recognition or ASR systems), or from scanned images (Handwriting Text Recognition or HTR systems). For Statistical Machine Translation (SMT) tasks the input sequence $\bar{x}$ is a sequence of words from the source language. All these systems are composed of two different stages (see Figure 1.1 on page 7): a supervised training stage where the statistical models are estimated; and an evaluation stage where the system outputs its best hypothesis and the system error could be evaluated.

Under the statistical pattern recognition framework, the fundamental equation [Duda *et al.*, 2001] is:

$$\hat{\bar{y}} = \arg\max_{\bar{y} \in \Omega^+} p(\bar{x}|\bar{y}) p(\bar{y}) \,. \tag{1.2}$$

This equation can be generalized as a log-linear combination of M information sources under the Maximum Entropy framework [Berger *et al.*, 1996a]:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} \prod_{m=1}^{M} \mathcal{H}_m(\bar{x},\bar{y})^{\lambda_m} , \qquad (1.3)$$

where $\mathcal{H}_m$ is the score (or probability) computed by the model m, and λ_m is the combination factor of the model m. For sequence recognition tasks, under the Hidden Markov Model framework, three models are combined: the Markov generative model probability, the Language Model (LM) probability, and a hypothesis length penalization. Therefore, three weights λ are needed. The Markov generative weight is always fixed to one ($\lambda_0 = 1$), the LM weight (λ_1) is known as Grammar Scale Factor (GSF), and the hypothesis penalization weight (λ_2) is known as Word Insertion Penalty (WIP). So, the equation for sequence recognition is:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y})^{\lambda_1} exp(|\bar{y}|)^{\lambda_2} . \qquad (1.4)$$

D.1.2. Dataflow architecture

All developed systems share a common modular architecture known as `dataflow`. Each `dataflow` is built of modules where almost every communication flows left-to-right (one way channels), and in some special cases right-to-left (feedback channel). Using the feedback channel, the algorithms implemented in the modules are synchronized, and they exchange best partial hypotheses probabilities for pruning purposes.

All the `dataflow` systems are formed by two semantic parts: hypothesis generation which builds a topological sorted lattice; and Viterbi decoding search, which incorporates LM probabilities and others, and extracts an N-best graph/list, or the single one-best hypothesis.

Incident graph protocol

The graph protocol defines a set of messages that are useful to the description of a directed acyclic graph in a topological order. This protocol is called "incident graph protocol" [España-Boquera *et al.*, 2007] because all incoming edges of a vertex are sent together. The full repertory of messages is described in Table 1.1 on page 11, and an example is shown in Figure 1.2 on page 10.

This protocol can be generalized to multistage graphs with the addition of new restrictions. In a multistage graph, the nodes are partitioned into stages, and incoming from a current stage node are forbidden. The multistage extension of the graph protocol is shown in Table 1.2 on page 12, and an example can be found in Figure 1.3 on page 13. Multistage graphs are related to machine translation problems, where the search is built over the source language reordering graph.

D.1.3. Evaluation measures

This work uses different evaluation measures depending on the task:

- Perplexity (PPL) [Brown *et al.*, 1992] will be used for language model comparison purposes.

- Word Error Rate (WER) useful for ASR and HTR tasks. The Character Error Rate (CER), Concept Error Rate (CER), or Sentence Error Rate (SER) WER extesions will be used depending on the task.
- Bilingual Evaluation Understudy (BLEU) [Papineni *et al.*, 2002] will be used in SMT tasks.
- Translation Edit Rate (TER) [Snover *et al.*, 2006] will be also used to SMT tasks.

D.1.4. Statistical significance of experimental results

We follow the bootstrapping technique described in [Koehn, 2004].

Confidence interval computation

Confidence interval estimation is done using the bootstrapping technique. A stochastic method iterated $1\,000$ times, that takes with replacement K random sentence pairs (reference, system output), was implemented. At each iteration, the evaluation measure is computed. Following, the $1\,000$ iterated evaluation measure computations are sorted, and the $5\,\%$ from the extremes ($2.5\,\%$ from each extreme) are released. The limits of the confidence interval are the remaining extremes on the ordered sequence. Figure 1.4 on page 18 shows a scheme of this procedure.

Pairwise comparison

Sometimes the confidence interval between two systems is lightly overlapped. Then, it is possible to do a *pairwise comparison* test that measures if the difference between the two systems is significant or not. This test is performed doing a confidence interval computation but, instead of computing the interval over an evaluation measure result, it computes the interval over the difference between the evaluation measure of each system.

D.1.5. Discriminative training of weight combination

There exists in machine translation a very extended procedure known as Minimum Error Rate Training (MERT) [Och, 2003]. It is an iterative method that estimates an optimum value for the weights of a log-linear combination. All the developed algorithms of this thesis use MERT to estimate the log-linear combination weights, even for sequence recognition to estimate the GSF and WIP (see Equation (1.4) on page 8eq:intro:gsfwip:1). This procedure alternates three basic steps:

1. Decode a development set using any set $\bar{\lambda} = \lambda_1 \lambda_2 \dots \lambda_M$ weights, obtaining an N-best list for each input sequence.
2. Optimize the log-linear combination weights using any optimization algorithm:
   - In machine translation, we will use the MERT implementation of Moses toolkit [Koehn et al., 2007], which uses the Powell optimization algorithm [Powell, 1964].
   - In sequence recognition we will use an in-house implementation of the MERT, using the Simplex optimization algorithm [Nelder & Mead, 1965].

This procedure generates a new set $\bar{\lambda}'$ of weights.

3. If the dissimilitude between $\bar{\lambda}'$ and $\bar{\lambda}$ is small enough, then stop. Else, substitute $\bar{\lambda}$ with $\bar{\lambda}'$ and repeat from step 1.

Figure 1.5 on page 20 shows an scheme of this algorithm. Currently, in the HTR scientific community, it is usual to optimize GSF and WIP using trial and error procedures. Then, one of the minor contributions of this thesis is to propose an extended use of MERT also in HTR for GSF and WIP optimization.

D.1.6. Scientific and technological goals

The scientific goals of this thesis are:

1. Formal specification of NN LMs as Stochastic Finite State Automata.

2. Specification of algorithms with totally coupled NN LMs into the decoding step.

3. Empirical study comparing NN LMs integrated into the decoding vs NN LMs in an N-best rescoring step.

4. NN LMs extension with an additional input module useful to do LM adaptation, taking into account long-term dependencies as a cache LM.

Technological goals are:

1. Implementation of efficient algorithms to train NN LMs.

2. Implementation of efficient algorithms to evaluate NN LMs.

3. Implementation of efficient algorithms to enable the use of NN LMs in sequence recognition and machine translation tasks, coupled into the decoder.

4. Use of the MERT algorithm to establish the combination weights on HTR tasks.

5. Contribution to the development of the `April` toolkit, which contains all the algorithms used on this thesis, and other implementations from other members of the research group.

Finally, note that interesting improvements were achieved in the addressed tasks:

- competitive results in SLU tasks;
- improvement of state-of-the-art performance in HTR;
- state-of-the-art performance in machine translation, as it was shown at the international evaluations of our developed systems (see appendix C).

D.2 Language models and NN LMs

D.2.1. Statistical language models

The Language Model (LM) was introduced for first time under the statistical pattern recognition framework using Equation (1.2) on page 8. The term $p(\bar{y})$ is the LM a priori probability of a given system hypothesis $\bar{y}$. Under the statistical approach, the LM probability is computed following Equation (2.1) on page 28:

$$p(\bar{y}) = \prod_{j=1}^{|\bar{y}|} p(y_j | y_{j-1} y_{j-2} \dots y_1) \,. \tag{2.1}$$

Using this formulation, the number of model parameters increases exponentially with the sentence length. The most extended simplification of this equation is to use an N-gram LM [Jelinek, 1997] where the conditional probability computation is reduced to the context of the previous $N-1$ words:

$$p(\bar{y}) \approx \prod_{j=1}^{|\bar{y}|} p(y_j | \bar{h}_j) \,, \tag{2.2}$$

where $\bar{h}_j$ is $y_{j-1} y_{j-2} \dots y_{j-n+1}$, the $N-1$ previous words (or prefix) of N-gram that ends at word y_j.

Language model quality can be measured with its perplexity (PPL) on a test set. The correlation between PPL and the final system error (measured as WER, BLEU, or TER) is very task dependent and is not totally clear. Some authors made studies of the relationship between PPL and WER [Iyer *et al.*, 1997; Chen *et al.*, 1998; Clarkson & Robinson, 1999; Ito *et al.*, 1999; Printz & Olsen, 2002; Klakow & Peters, 2002], concluding that the correlation is hidden by the test set characteristics, nevertheless the correlation exists. As an illustrative example, Figure 2.1 on page 30 shows the evolution of WER vs PPL on the IAM task [Marti & Bunke, 1999] of HTR and the French Media task [Bonneau-Maynard *et al.*, 2005] of ASR. That figure illustrates that exists a correlation, but some noise on the plot due to the characteristics of the test set also exists. Nevertheless, if the reduction of PPL is high enough, a reduction on WER could be ensured.

D.2.2. N-gram language models

N-grams [Jelinek, 1997] are the simplest and most extended kind of LM. N-grams are a good trade-off between system performance and result quality, but they have some deficiencies. The complexity of the model increases exponentially with the order value N. Even with large training textual corpus, there exists an exponential number of N-grams which does not appear in the training material, and their parameters cannot be estimated. Several solutions to this problems exist in the literature [Ney & Essen, 1991; Chen & Goodman, 1996]. The NN LMs [Bengio *et al.*, 2003; Bengio, 2008] can deal with this problems because the projection of words into a continuous space permits the neural network to estimate N-gram probabilities with some kind of non-linear interpolation on this projection space.

The basic formulation to estimate the conditional probabilities of an N-gram LM is based on word and N-gram counts. Let $\bar{k}$ an N-gram which $\bar{k} = \bar{q}x$, being $\bar{q} \in \Omega^{N-1}$ the

N-gram prefix (or context) and $x \in \Omega$ the last word of the N-gram. The probability $p(x|\bar{q})$ is computed as:

$$p(x|\bar{q}) = \frac{\mathcal{C}(\bar{q}x)}{\mathcal{C}(\bar{q})} ,$$

being $\mathcal{C}(\bar{q}x)$ the count of the N-gram formed by $\bar{q}x$, and $\mathcal{C}(\bar{q})$ the count of the $N-1$ length prefix of the N-gram. Note that $\mathcal{C}(\bar{q}) = \sum_{\omega \in \Omega} \mathcal{C}(\bar{q}\omega)$.

This basic formulation was modified and extended in the literature to deal with N-gram deficiencies. Smoothing techniques [Lidstone, 1920; Johnson, 1932; Jeffreys, 1948; Good, 1953; Jelinek & Mercer, 1980; Brown *et al.*, 1992; Witten & Bell, 1991] are needed to redistribute the probability mass of N-grams that are unseen on training material. Back-off techniques [Kneser & Ney, 1995] compute the N-gram conditional probability as a combination of a back-off smoothing probability and an N-gram of lower order. Interpolation of different order N-grams is useful to generalize the probability computation.

The selected smoothing technique in this work for standard N-grams is the interpolated back-off *modified Kneser-Ney* model [Kneser & Ney, 1995]. This technique is one of the most extended in the literature and shows performance improvements over other techniques.

D.2.3. Connectionist language models

NN LMs are a kind of N-gram language models trained using artificial neural networks (ANNs). ANNs has two major benefits to compute LM probabilities: the words are projected into a continuous space, computing a distributed encoding of each vocabulary word; and N-gram probabilities are computed interpolating the input concatenation of the $N-1$ context word projections, which permits ANNs to learn automatically some non-linear smoothing for unseen N-grams.

The idea of encoding distribution as input for ANNs comes from [Hinton, 1986; Elman, 1990; Paccanaro & Hinton, 2000] and in language modelling from [Nakamura & Shikano, 1989; Miikkulainen & Dyer, 1991; Castro-Bleda *et al.*, 1999; Xu & Rudnicky, 2000]. The idea of a coupled train of the distributed encoding and N-gram probabilities using ANNs comes from [Bengio *et al.*, 2003; Schwenk, 2007].

A NN LM is formed by four layers (see Figure 2.2 on page 35):

- An input layer $\bar{I}$ containing $(N-1) \cdot |\Omega|$ neurons, where each group j of $|\Omega|$-size neurons is the j-th locally encoded input word ($\bar{L}_j$).

- Each $\bar{L}_j$ is connected with a group $\bar{P}_j$ which are the continuous space projection of the corresponding input word j. The weights matrix ($\mathbf{W}_{L,P}$ and bias vector $\bar{B}_P$) between each word j and its corresponding $\bar{P}_j$ are shared. All projections are concatenated to form the projection layer $\bar{P}$. Probabilities computation are done over this projection layer instead of over the *raw* locally encoded words. This layer is removed after training because for each word exists an unique distributed encoding (see Figure 2.3 on page 36).

- The projection layer is connected to a hidden layer $\bar{H}$ using the weights matrix $\mathbf{W}_{P,H}$ and the bias vector $\bar{B}_H$. The *tanh* activation function is used on this layer.

- Finally the hidden layer is connected to the output layer $\bar{O}$ throught the weights matrix $\mathbf{W}_{H,O}$ and the bias vector $\bar{B}_O$. This layer uses the *softmax* activation function which ensures a posteriori probability learning of the ANN [Bishop, 1995].

The NN LM forward computation, using vector-matrix notation, is:

$$\bar{P} = \bigcup_{i=1}^{N-1} \left(\bar{L}_i \cdot \mathbf{W}_{L,P}^T + \bar{B}_P \right) , \tag{2.17}$$

$$\bar{H} = tanh(\bar{P} \cdot \mathbf{W}_{P,H}^T + \bar{B}_H) , \tag{2.18}$$

$$\bar{A} = \bar{H} \cdot \mathbf{W}_{H,O}^T + \bar{B}_O , \tag{2.19}$$

$$\bar{O} = \frac{exp(\bar{A})}{\sum_{i=1}^{|A|} exp(a_i)} , \tag{2.20}$$

where a_i is the i-th component of vector $\bar{A}$. Due to the locally encoding of each $\bar{L}_i$, Equation (2.17) consists on an addition of the bias $\bar{B}_P$ and the column of $\mathbf{W}_{L,P}$ corresponding to the input word. The ANN is trained using the error Backpropagation (BP) [Rumelhart *et al.*, 1986] on its stochastic and on-line version (see Algorithm 2.1 on page 38) using the cross-entropy objective function (see Equation (2.21) on page 37 and Equation (2.22) on page 37). The algorithm uses a random replacement procedure [Schwenk, 2007]. To avoid overfitting, the L_2 regularization term (weight decay) is added to the objective function (see Equation (2.23) on page 37).

The ANN training procedure could be summarized as follows:

$$\frac{\partial E}{\partial O} = \left\langle o_1 - t_1, o_2 - t_2, \ldots, o_{|\bar{O}|} - t_{|\bar{O}|} \right\rangle , \tag{D.1}$$

$$\bar{B}_O = \bar{B}_O + \eta \frac{\partial E}{\partial O} , \tag{D.2}$$

$$\mathbf{W}_{H,O} = \epsilon \mathbf{W}_{H,O} + \eta \bar{H}^T \frac{\partial E}{\partial O} , \tag{D.3}$$

$$\delta \bar{H} = \mathbf{W}_{H,O} \frac{\partial E}{\partial O} , \tag{D.4}$$

$$\frac{\partial E}{\partial H} = \left\langle \delta \bar{h}_1 \cdot 0.5(1 - \bar{h}_1^2), \ldots, \delta \bar{h}_i \cdot 0.5(1 - \bar{h}_i^2), \ldots \right\rangle , \tag{D.5}$$

$$\bar{B}_H = \bar{B}_H + \eta \cdot \frac{\partial E}{\partial H} , \tag{D.6}$$

$$\mathbf{W}_{P,H} = \epsilon \mathbf{W}_{P,H} + \eta \cdot \bar{P}^T \cdot \frac{\partial E}{\partial H} , \tag{D.7}$$

$$\frac{\partial E}{\partial P} = \mathbf{W}_{P,H} \cdot \frac{\partial E}{\partial H} , \tag{D.8}$$

$$\bar{B}_P = \bar{B}_P + \frac{1}{N-1} \sum_{i=1}^{N-1} \eta \frac{\partial E}{\partial P_i} , \tag{D.9}$$

$$\mathbf{W}_{L,P} = \epsilon \mathbf{W}_{L,P} + \frac{1}{N-1} \sum_{i=1}^{N-1} \eta \bar{L_i}^T \frac{\partial E}{\partial P_i} . \tag{D.10}$$

Usually, the weight decay term is ignored in the projection layer, sometimes to ensure that weights of *rare* words are not zero. The bias vector is also ignored in the projection layer. However, it could be useful to combine bias and weight decay to ensure that *rare* words has a common encoding [Hai-Son *et al.*, 2010]. In this thesis, we use the bias vector and the weight decay term in the projection layer to help the achievement of a common encoding of non-frequent words.

D.2.4. NN LMs and standard N-grams combination scheme

In order to enhance the generalization power of neural networks, the used NN LMs will be a combination of some neural network outputs that differ in the projection layer size [Schwenk, 2007]. To fulfill the same goal, log-linear or linear combinations will be done with standard N-grams. In the literature these two combination schemes achieve similar performance [Schwenk & Koehn, 2008].

Depending on the task, models will be combined linearly or log-linearly. Let be a set of K neural networks to be combined, denoting with $p_k(y_j|\bar{h}_j)$ for $k = 1, 2, \ldots, K$ the probability computed by the neural network k, and denoting with $p_0(y_j|\bar{h}_j)$ the probability computed by the standard N-gram. Usually the standard N-gram is computed over the whole task vocabulary and with the interpolated modified Kneser-Ney smoothing method. The combination of the neural network models and standard N-grams will be:

- Linear combination: all the neural networks plus the standard N-gram are linearly interpolated. Each model has a combination coefficient α_k, which is estimated to minimize the PPL on a development set. The combination follows Equation (2.36) on page 41:

$$p(y_j|\bar{h}_j) = \sum_{k=0}^{K} \alpha_k p_k(y_j|\bar{h}_j)\,. \tag{2.36}$$

  This scheme will be the prefered on ASR and HTR tasks.

- Log-linear combination: in this case, the neural networks will be linearly combined, and the resulting model will be log-linearly combined with the standard N-gram. The α coefficients will compose the linear combination of neural networks, λ_0 will be the log-linear combination coefficient of the standard N-gram, and λ_1 the corresponding log-linear coefficient for the linearly combined neural network N-gram model. In this case, the log-linear combination will be estimated inside the Maximum Entropy Equation (1.3), minimizing the BLEU:

$$p(y_j|\bar{h}_j) = p_0(y_j|\bar{h}_j)^{\lambda_0} \left(\overbrace{\sum_{k=1}^{K} \alpha_k p_k(y_j|\bar{h}_j)}^{\text{linear combination}} \right)^{\lambda_1}\,. \tag{2.37}$$

  This scheme will be the prefered on SMT tasks.

D.2.5. NN LM deficiencies

One of the major contributions of NN LMs is the distributed encoding of words at the projection layer. This encoding is position independent, and allows the neural network to interpolate unseen input word sequences. However, this encoding has several problems with words with low frequency counts [Hai-Son *et al.*, 2010]. Sometimes, with large textual training data, these words could not be showed to the neural network, leading to a poor encoding of these *rare* words. Enhancing the quality of this encoding normally leads to some minor improvements.

The second major problem of NN LMs is their computational cost on the evaluation stage. This cost is dominated by the softmax output layer [Schwenk, 2007], because it forces the computation of *all* N-gram probabilities given the input, although only a few are needed. In this thesis, a novel idea to deal with this computational cost will be presented, empirically evaluated, and discused. This reduction will make possible to develop a totally coupled decoding algorithm.

D.3 Contributions to connectionist Language Modeling

This section summarizes the major contributions of this work to the connectionist Language Modeling field.

D.3.1. Encoding of words with low frequency counts

As it was previously commented, words with low frequency counts are a problem due to their poor trained encoding [Hai-Son *et al.*, 2010]. This work will adopt a simple approach based on the use of a restricted input vocabulary on the NN LM. This new vocabulary Ω^I will be a subset of the whole vocabulary Ω of the task. An input neuron will be added to take into account the class of *rare* words out of Ω^I.

This problem was discused by other authors [Hai-Son *et al.*, 2010], trying different initialization and training schemes of the projection layer. They conclude that it is best to initialize the encoding of all the input words as a unique random value. Under this scheme, *rare* words change a little bit (or not change at all) their initial encoding, so, at the end, all *rare* words share the same encoding. Probably, weight decay parameter, combined with bias vector, could be used on the projection layer to obtain a similar behaviour, but authors only use this two parameters on the hidden and output layers.

D.3.2. Computational cost reduction

Similar computational problems appear on the use of NN LMs at both training and evaluation phases. The computation of the softmax normalization constant is a major concern because it forces the computation of every output word, even when only a few words are necessary. At the evaluation phase, this problem is even more important.

Nevertheless, other computational issues have to be solved, and this work implements the following improvements.

32 bits float precision

Float numbers in NN LMs implementation is only for single precision (32 bits), which has been demonstrated to be enough to train neural networks [España-Boquera *et al.*, 2007a; Schwenk, 2007]. Memory locality due to single precision numbers versus double precision numbers speeds-up aritmethic operations.

Restricted vocabulary at the neural network output layer

A reduced output layer means a drastical reduction of computational cost. This layer is the most time consuming, and the application of the Short-List idea [Schwenk, 2007] reduces its size to only the 10K or 20K most frequent words. Short-List vocabulary will be denoted by Ω'. The rest of words will be denoted as $OOS = \Omega - \Omega'$. Some approaches define a special OOS output neuron that computes the probability mass of all words out of the Short-List.

OOS words probability mass will be smoothed using a trade-off solution between the following ideas:

- Schwenk [2007]: uses a standard N-gram to smooth OOS words, following Equation (3.1) on page 48, or Equation (3.2) on page 48.
- Emami & Mangu [2007]: defines the NN LM mass probability for OOS words to be zero, following Equation (3.3) on page 49.
- Park *et al.* [2010]: makes the assumption that the probability mass computed by the NN LM for the OOS neuron is equal to the probability mass computed using a standard N-gram, following Equation (3.4) on page 49.

In this thesis, a simple unigram computed over the OOS words will be used to make the smoothing of the OOS words probability, following Equation (3.5) on page 49, reproduced here:

$$p(y_j|\bar{h}_j) = \begin{cases} p_{NN}(y_j|\bar{h}_j), & \text{iff } y_j \in \Omega' \,; \\ p_{NN}(OOS|\bar{h}_j) \cdot \dfrac{\mathcal{C}_{OOS}(y_j)}{\sum\limits_{y' \notin \Omega'} \mathcal{C}_{OOS}(y')}, & \text{iff } y_j \notin \Omega' \,, \end{cases} \qquad (3.5)$$

being $p_{NN}(\omega|\bar{h}_j)$ the probability of word ω given the previous history $\bar{h}_j$ computed using the neural network. Note that an additional output unit that computes the probability of OOS class is needed.

Bunch mode

The bunch mode consists in substituting the online learning algorithm by a mini-batch learning algorithm, where the mini-batch size defines a bunch of patterns that could be learned simultaneously using matrix-matrix operations [Schwenk, 2007]. Using CBLAS API, which offers a C implementation of BLAS API [Lawson *et al.*, 1979; BLAS, 1979], the computation of matrix-matrix operations could be accelerated by means of SSE and vectorial CPU operations. During the evaluation phase, it is possible to use the bunch mode launching the forward of several histories $\bar{h}_j$.

Speed-up techniques for the evaluation phase

Some especific techniques could be applied to speed-up the evaluation phase:

- A structured hierarchy decomposition of the output layer, obtaining a binary and stochastic decision tree. This approach could reduce the $O(N)$ cost of the softmax normalization to $O(\log N)$ [Morin & Bengio, 2005; Hai-Son *et al.*, 2011], being N the vocabulary size.

- A novel idea is presented as a contribution of this thesis. Precompute and store in a table the most important softmax normalization constants will reduce the computation cost of stored constants to $O(1)$. In this way, it's not necessary to compute the whole NN LM output, only LM look-ups generated by the decoding process will be computed. When a constant is not found, it can be computed *on-the-fly* or any kind of smoothing can be applied. This approach will be deeper described in Section D.4.

D.3.3. Experiments using different input vocabulary sizes

Some experiments testing different input vocabulary sizes ($|\Omega^I|$) were conducted on two different tasks. Results using different N order and different $|\Omega^I|$ values are shown in Table 3.1 on page 52:

- IAM-DB HTR task: medium size vocabulary task with $|\Omega| = 103$K words. No significative differences were observed for the tested input vocabulary sizes.

- BTEC'06 Italian-English SMT task: small vocabulary task with $|\Omega| = 7.2$K words. No differences were observed on PPL, but yes on BLEU.

Observing these results, it's not clear the effect of the input vocabulary size on the quality of the model. Nevertheless, theoretically, a bigger input vocabulary will cover a larger number of N-grams histories, but it will have a deficient encoding of *rare* words. A deeper research is needed to solve clearly this issue.

D.3.4. NN LMs integration in the decoding systems

The NN LM, as an N-gram LM, can be represented using Stochastic Finite State Automatas (SFSAs) [Llorens Piñana, 2000]. In this way, the decoding algorithm is provided with an automata state identifier that represents which context $\bar{h}_j$ was seen until the current instant. This identifier will be denoted by LMkey, and it is updated with each LM transition. The LMkey represents the sequence of $N-1$ words needed to arrive to the current automata state. Figure 3.1 on page 53 shows a small example of an N-gram model represented as a SFSA.

Taking into account the LMkey, algorithms related with LMs will work over a generalization of the N-gram LM concept, using an abstract C++ interface showed in Table 3.2 on page 54.

NN LMs as a stochastic finite state automata

A NN LM is a compact form representation of a full N-gram LM, and consequently it is a compact form representation of a SFSA. A NN LM represents the set of $|\Omega|^N$ possible N-grams for a given vocabulary Ω.

Due to the huge size of the resulting automata, it is not expandible in memory, so, an on-the-fly approach will be followed: initially the automata only contains the initial LM state, the rest of states will be enumerated with the new incoming hypothesis. Each automata state will be identified by a unique LMkey, that represents the sequence of $N-1$ context words related to the state.

An auxiliar Trie data structure, denoted by TrieLM, will represent sequences of N-grams. The TrieLM is a static hash table with open addressing. A LMkey will be a position on the hash table. Therefore, given a LMkey, the TrieLM could be asked by the sequence of $N-1$ words which are related with it. The TrieLM has a persistent part (used to store LMkeys related with precomputed softmax normalization constants), and a dynamic part (used during decoding to generate on demand new LMkey identifiers). Figure 3.2 on page 55 shows an example of TrieLM. Each TrieLM node contains:

- A parent index.
- A timestamp value useful to implements the clear operation on the hash table. The timestamp 0 is reserved to persistent states.
- A transition word.

The TrieLM implements a generic interface described in Table 3.3 on page 56.

NN LM decoding algorithms

Using the TrieLM and the LM interface, it is possible to implement generic Viterbi algorithms that are independent of the N-gram LM type (standard N-gram or NN LM). Using this approach, and the softmax normalization constants precomputation (described deeply at Section D.4), the computational issues of the NN LMs will be reduced to an asumible cost.

It is possible to implement an N-best lists rescoring system using the TrieLM, taking profit of the bunch mode and other speed-up ideas [Schwenk, 2007]. Algorithm 3.1 on page 57 describes the implementation of the N-best lists rescoring system using the TrieLM. The bunch mode is not detailed in the algorithm, but it could be extended in a straightforward way. Figure 3.3 on page 59 shows a flow diagram of the N-best lists rescoring approach.

D.3.5. Other proposals of improvement

- NN LMs extended to work as a cache LM: motivated by Kuhn & Mori [1990], the basic idea is to introduce a new input layer to take into account long distance context information.
- Projection layer initialized with syntactic POS information: one initial encoding of the input words could be randomly selected for each possible POS tag. Words initial encoding could be computed combining the conditional probability of the POS tags given the word, and the initial encoding of each tag.

- Adding momentum term: this term is useful in some machine learning tasks, leading to a smoothed learning curve. The momentum computation cost will be $O(2|\mathbf{W}|)$, being $\mathbf{W}$ the weights matrix. A training algorithm description using momentum and weight decay is shown in Algorithm 3.2 on page 61.
- Generic Linear Interpolation (GLI) in continuous space: GLI [Hsu, 2007] seems very interesting to enhance the combination of NN LMs and standard N-grams. Instead of doing a simple linear interpolation, a neural network could compute the interpolation value of LMs given their context history $\bar{h}_j$, as shown in Equation (3.6) on page 62. The context history could be represented as a sequence of words in a continuous space. A preliminar experiment shows the future posibilities of GLI using neural networks, as it is shown in Figure 3.4 on page 64.

D.3.6. Summary

This section describes briefly some contributions of this thesis to connectionist language modeling:

- Formalization of NN LMs totally integrated into the decoding algorithms.
- Some techniques for computational cost reduction were presented. The most important will be detailed in the next section.
- The idea of extended NN LMs with an input cache layer was presented, and it will be deeply described in section D.5.

Three interesting future work lines were commented:

- A method to initialize the continuous space projection of words using syntactic POS information.
- Addition of a momentum term to the learning algorithm.
- Use of Generalized Linear Interpolation to improve the combination between NN LMs and standard N-grams. The GLI could be computed using a neural network and the continuous space projection of words.

D.4 Fast evaluation of NN LMs

This section summarizes the algorithm to speed-up the evaluation phase of NN LMs. The major part of this section was published at [Zamora-Martínez *et al.*, 2009], therefore this text only summarizes results different from the previous published work.

The goals and contributions of this section are:

- Formalization of the speed-up technique used for softmax normalizacion, published at [Zamora-Martínez *et al.*, 2009].
- Integration of NN LMs in a HTR task.
- Evaluation of quality and performance comparing standard N-grams, standard NN LMs, and the proposed approach.

D.4.1. speed-up technique for softmax normalization constants

Details about the speed-up technique are published at [Zamora-Martínez *et al.*, 2009]. Some Equations can be found in chapter 4. The computation of the log-probability of the NN LM output, using the softmax constant Z_I, given the input $\bar{I}$, is shown at Equation (4.3) on page 70, and reproduced here:

$$\log Z_I = \log \sum_{j=1}^{|\Omega|} exp(a_j) \tag{4.1}$$

$$\log o_i = \log \frac{exp(a_i)}{Z_I} = a_i - \log Z_I \tag{4.2}$$

$$\log p(\omega_i|\bar{h}_i) = \log o_{\omega_i}, \tag{4.3}$$

Each normalization constant is stored in a hash table using as index the TrieLM node (or LMkey) corresponding to the NN LM input sequence of words. Algorithm 4.1 on page 72 describes the method **`prepareLM`** that uses the speed-up technique with a hierarchy of models.

To ensure an efficient integration of NN LMs at the decoding phase, two intermediary tables (or cache tables) were introduced:

1. N-grams probability cache: it associates the pair $(\text{LMkey}, \omega \in \Omega)$ with the probability $p(\omega|h)$. It is a big cache, with 512 thousand entries.

2. Hidden units cache: it associates an LMkey with the hidden units activation. This is a medium cache, with 32 thousand entries.

D.4.2. Perplexity evaluation experiments

See [Zamora-Martínez *et al.*, 2009].

D.4.3. Experiments on NN LMs integrated into the decoding phase

This section summarizes experimentation results using the HTR system with integrated NN LMs.

Smoothed Fast NN LM integration effect

Figure 4.3 on page 79, Figure 4.4 on page 79, and Figure 4.5 on page 80 show the effect of histogram pruning on the WER and computational cost. These results compare the best standard N-gram and the best NN LMs found for this task. The standard N-gram is a bigram, and the NN LMs are a smoothed Fast NN LM which combines bigrams, 3-grams, and 4-grams on a hierarchy of models to speed-up the evaluation of NN LMs. Also, results with on-the-fly approach[1] are shown. Three different perspectives were used to compare this results.

[1]The same as a standard NN LM but using softmax normalization constant precomputation to speed-up the decoding. The constants that are not found were computed on-the-fly and stored during current sentence decoding.

1. **Relationship between WER and time:** Figure 4.3 on page 79 shows a point where smoothed Fast NN LMs presents a better computational cost than standard N-grams, given the same WER results. In any case, the WER is always better for the smoothed Fast NN LM (see Figure 4.5 on page 80). The standard NN LM (using on-the-fly approach) shows a computational cost 7 times higher.

2. **Relationship between WER and histogram pruning size:** Figure 4.4 on page 79 compares the WER improvement achieved by the smoothed Fast NN LM given a histogram pruning value. In the best case, the WER improvement is $\approx 8\,\%$, an absolut difference of 1.7 points. Differences between standard NN LM and smoothed Fast NN LM are not statiscally significative.

3. **Computational cost of smoothed Fast NN LMs and standard NN LMs:** Figure 4.5 on page 80 compares the computational cost for different histogram pruning values. For the strongest pruning (100 active states), the smoothed Fast NN LMs need a $\approx 50\,\%$ more time than standard N-grams. This difference is reduced constantly with the increasing number of histogram pruning values. For the weakest pruning ($10\,000$ active states) the difference is of a $\approx 5\,\%$ more time. This plot shows a big difference between smoothed Fast NN LMs and standard NN LMs. The standard NN LM system is almost two times slower, which suposes a $\approx 100\,\%$ cost increase.

Conclusions

The smoothed Fast NN LMs introduces a little increase in computational cost, compared with standard N-grams. This increase is a trade-off between quality and velocity, and in every case the smoothed Fast NN LM approach shows better WER results (between a $5\,\%$ and $50\,\%$ slower). However, for a given WER system performance, it is possible to build a system using smoothed Fast NN LMs that works faster than the standard N-gram system.

D.4.4. Summary

The technique used for speed-up evaluation of NN LMs were presented and empirically tested. The achieved goals are:

- Formalization of the speed-up technique to accelerate the computation of softmax normalization constants.

- Empirical comparison of this technique with standard N-gram approach. This result was published in [Zamora-Martínez *et al.*, 2009], achiving a speed-up of 33 comparing smoothed Fast NN LMs with standard NN LMs.

- Empirical evaluation of NN LMs totally coupled into the decoding phase for a HTR task. Impressive WER results (an improvement of 1.7 absolut points), and almost identical computational costs were achieved. A journal paper is being prepared to present these results in the "IEEE Transactions on Neural Networks" journal.

D.5 NN LMs with cache

This section describes the NN LM extension for language model adaptation through a cache input layer, based on the ideas of cache LMs [Kuhn & Mori, 1990]. This new model will be denoted as Cache NN LM.

The paper [Zamora-Martínez *et al.*, 2012a] describes almost everything of Chapter 5 on page 83, therefore this text only relies on details that couldn't be found at the paper. It is mandatory to read sections 1, 2, and 3 of the paper for a full understanding of the notation and details of this section.

Figure 5.1 on page 87 shows an example of a NN LM extended with the new cache input layer. This input cache is a summary of all previous sentences related with current sentence and document decoding, and denoted by $\bar{h}_1^{u-1}$, being u the index of the current sentence at the current document. Following Equation (5.3) on page 87, the decoding using Cache NN LM is straightforward.

The parameter $\bar{h}_1^{u-1}$ is codified on the neural network using the feature vector $\bar{C}$. Each component of the feature vector is an input neuron of the cache layer. $\bar{C}$ is a bag-of-words vector, that is, each component represents a word, and its activation means the presence of its related word at the parameter $\bar{h}_1^{u-1}$. Word activations are exponentally decayed on each new processed input word.

D.5.1. Context codification for the cache

It is possible to distinguish three kinds of information sources where to extract data for the cache parameter $\bar{h}_1^{u-1}$:

- **Known data:** events really known. For instance, in a SMT system, source language sentences are information of this type.
- **Probably erroneous data:** sentences decoded by the system.
- **Data with dependencies:** useful in dialog systems. For instance, machine sentences generated by a dialog system to answer user interactions. These sentences are grammatically correct, but the system could missunderstand user intentions.

Algorithm 5.1 on page 89 shows how to generate the feature vector $\bar{C}$ on-the-fly in order to train a Cache NN LM.

D.6 Hidden Markov Models and decoding

This section summarizes the most important aspects of the HMM decoder implemented for ASR and HTR tasks. Details of this algorithm were published in [España-Boquera *et al.*, 2007; España-Boquera *et al.*, 2007b, 2011]. Under the pattern recognition approach, HMM decoding follows Equation (1.2) on page 8, reproduced here:

$$\hat{\bar{y}} = \underset{\bar{y}\in\Omega^+}{\arg\max}\ p(\bar{y}|\bar{x}) = \underset{\bar{y}\in\Omega^+}{\arg\max}\ p(\bar{x}|\bar{y})p(\bar{y})\,. \tag{1.2}$$

However, in practice, it is generalized to introduce a scaling factor for the language model (Grammar Scale Factor or GSF), and a hypothesis length penalization (Word Insertion Penalty or WIP). Therefore, HMM decoding could be formalized under the maximum entropy approach following Equation (1.4) on page 8:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y})^{\lambda_1} exp(|\bar{y}|)^{\lambda_2} , \qquad (1.4)$$

being λ_1 the GSF applied to the LM, and λ_2 the WIP factor applied to an exponential modellization of hypothesis length. The value of these factors can be estimated by means of the MERT procedure, using Simplex [Nelder & Mead, 1965] as optimization algorithm.

D.6.1. HMMs hybrized with Neural Networks

The approach followed in this work differs from standard HMMs in the emission probabilities computation. HMMs hybrized with ANNs (HMM/ANN models) were used instead of gaussian mixture models. This approach show some advantages [Bourlard & Morgan, 1994; Bishop, 1995; España-Boquera *et al.*, 2011]. The probabilities computed by neural networks need to be transformed following Bayes rule (see Equation (6.2) on page 98) to convert posteriors in generative probabilities. Figure 6.2 on page 98 shows an example of the developed architecture. HMM/ANN models were trained following an Expectation-Maximization (EM) algorithm, with Viterbi segmentation and HMM transition probabilities estimation on the Expectation step, and neural network training on the Maximization step.

D.6.2. Decoding algorithm architecture

The implemented algorithm is a modification of the two-level algorithm [Sakoe, 1979] to take into account a global criteria during beam search. The algorithm consists in these two major steps:

a) Word graph generation from sequence of frames $\bar{x}$. This word graph is computed using HMM/ANN models and a tree lexicon based decoder [España-Boquera *et al.*, 2007b], called ViterbiM. The search is synchronized with next module using `score_feed_back dataflow` messages. Returned information is used for beam search enhancement.

b) Viterbi decoding using a LM over the word graph. Every time all incoming edges of a vertex were processed, the algorithm stops and sends to the previous module the best active state score by means of message `score_feed_back`.

D.6.3. Dataflow modules and algorithms

The whole algorithm is formed by several subprocesses on the `dataflow` architecture. The processes or modules are conected between themselves. Figure 6.3 on page 100 shows a diagram of the whole architecture, composed by these modules:

- Feature vectors source: **FrameSource**. This module could send frames from a microphone, a disk file, a socket, ...
- Generative HMM probabilities computation: **GenProbs**. Uses an ANN and applies the Bayes rule.

- Oversegmention, which controls the generation of new vertices between frames (true or false frontiers) on the word graph: **OSE**. Depending on the task, this module could be: a dummy (generates always a true frontier) on ASR tasks; a connected compound based algorithm (generates a true frontier on the limits of the compound) for HTR tasks.
- Word graph generator given the frames and the frontiers: **WGen**. It uses the ViterbiM algorithm and a pool of parsers to generate the vertices and edges of the word graph. Each parser was labeled with the vertex where it was created. Algorithm 6.1 on page 103 describes what it does when a true frontier. Algorithm 6.2 on page 104 describes the procedure executed with every frame.
- N-grams Viterbi algorithm over the search word graph: **NGramParser**. It uses an N-gram LM and the input word graph to obtain the one-best, a list of N-bests hypotheses, or an output lattice. This module executes different algorithms, reacting to `dataflow` messages. The `edge` procedure is described at Algorithm 6.3 on page 105.

Figure 6.4 on page 106 shows an example of the decoding process, showing an input image, the corresponding frontiers between frames, and the edges generated between true frontiers (each true frontier is a word graph vertex). Finally, the trellis corresponding for the **NgramParser** was shown.

D.7 Automatic Speech Recognition and Spoken Language Understanding

This section summarizes the experiments performed with the French Media task. Section goals are:

- Present a baseline ASR system with state-of-the-art performance.
- Extension of the baseline ASR system for SLU tasks.
- Improve SLU performance by means of Cache NN LMs.

The most important introduction and results of this section were published at ICASSP'12 [Zamora-Martínez *et al.*, 2012a]. Therefore, this section only relies on some figures and tables not included there.

D.7.1. French Media corpus

See beginning of section 4 of [Zamora-Martínez *et al.*, 2012a] for a full description of this SLU task. Figure 7.4 on page 114 and Figure 7.5 on page 115 show an example of this task.

D.7.2. Acoustic models

HMM/ANN acoustic models, with a MLP for the emission probabilities estimation, are used. Details of topology and parameters of the acoustic models are shown at Table 7.1 on page 116 and section 4 of [Zamora-Martínez *et al.*, 2012a].

D.7.3. SLU by using Language Models

The basic idea consists in the concatenation of words with its corresponding semantic concept tag, building a sequence of (concept,word) pairs. A LM is trained over this new corpus. Figure 7.6 on page 117 shows an example of a Media sentence and its corresponding sequence of concepts and supports. Under this approach, the search is reformulated as:

$$(\hat{\bar{c}}, \hat{\bar{y}}) = \underset{(\bar{c},\bar{y})\in\Omega_t^+}{\arg\max}\ p(\bar{c}, \bar{y}|\bar{x}) = \quad \text{D.11}$$

$$= \underset{(\bar{c},\bar{y})\in\Omega_t^+}{\arg\max}\ p(\bar{x}|\bar{c}, \bar{y})p(\bar{c}, \bar{y})\,, \quad \text{7.2}$$

where $\bar{c} = c_1 c_2 \ldots c_{|\bar{y}|}$ is the sequence of concept tags, each one associated with only one word of $\bar{y}$, and Ω_t is the vocabulary of (concept,word) pairs. Figure 7.7 on page 121 illustrates the scheme of the SLU whole system.

D.7.4. Experimentation

A full description of the experiments can be found in [Zamora-Martínez *et al.*, 2012a]. Nevertheless, all related tables are on the Spanish text of this thesis:

- Table 7.2 on page 120 and Table 7.3 on page 122 show ASR baseline results, using word-based LMs. The baseline is competitive with state-of-the-art results published by Hahn *et al.* [2011] ($30.3\,\%$ WER on validation and $31.4\,\%$ WER on test).
- Table 7.4 on page 123 shows the effect of (concept,word) pairs LMs on the WER of the final system.
- Table 7.5 on page 124 shows error of concepts (CER) of the final system.
- For comparison purposes, Table 7.6 on page 124 provides the best published results on the French Media SLU task.

D.7.5. Summary

The use of a cache memory model is proposed in a connectionist LM which captures long term dependencies of both words and concepts. Experiments conducted on a SLU task for a spoken dialog system have shown a significant CER reduction using a rather simple model. An effective SLU module should impose semantic coherence in searching a graph of (concept,word) hypotheses, while the results reported here refer only to the 1-best hypothesis obtained with a fairly simple acoustic model. An oracle error rate for the 1-best was also computed as the percentage of concepts appearing in the manual annotation but not in the sequence obtained with different LMs. Oracle values of $15.9\,\%$ for the best N-gram baseline and $14.2\,\%$ for the best cacheNNLM suggests that there is plenty of room for improvement. Future work will attempt to improve the HMM/ANN acoustic models and to conceive a new SLU component acting on (concept,word) pairs in confusion networks.

D.8 Handwritten Text Recognition

This section presents the full integration of NN LMs in a Handwritten Text Recognition (HTR) system. Different approaches were tested:

- Word-based NN LMs fully integrated into the decoding. A paper was submitted to the Pattern Recognition journal [Zamora-Martínez *et al.*, 2012c].
- Character-based NN LMs, published in [Zamora-Martínez *et al.*, 2010].
- Recurrent neural network LMs (based on Long-Short Term Memory Neural Networks) published in [Frinken *et al.*, 2012]. This is a collaborative work done with members of the IAM at the University of Bern.

The baseline HTR system was described in [España-Boquera *et al.*, 2011], where data processing and optical model details are explained (see sections 1-4).

D.8.1. Neural Network Language Models in HTR tasks

See [Zamora-Martínez *et al.*, 2012c], section 5.

D.8.2. Word-based NN LMs experiments

The characteristics of the data used for training language models are shown in Table 8.3 on page 138. Only a few notes to understand the experiments:

- Bigrams, 3-grams, and 4-grams were trained using the SRI toolkit [Stolcke, 2002], as a baseline for comparison purposes.
- NN LMs are a combination of the standard bigram model and four neural networks (projection layer sizes of 160, 192, 224, and 256 neurons; hidden layer of 200 neurons). A Short-List of the 10K more frequent words was used at the output layer, using Equation (3.5) for the computation of OOS words. Different input vocabulary sizes were tested, using the words that occur more than Θ times at training material, with $\Theta = 21, 10, 8, 1$, corresponding to 10K, 17K, 19K and 56K word vocabularies Ω^I. Bigrams, 3-grams and 4-grams NN LMs were trained.
- Experiments are denoted as follows:
  - $\Theta = 20$**K mKN**: the bigram LM used by Graves *et al.* [2009].
  - **mKN**: standard N-gram model trained with SRI toolkit.
  - $\Theta = v$ **NN**: standard NN LM, where v indicates the Θ value.
  - $\Theta = v$ **FNN**: smoothed Fast NN LM, using the fast evaluation technique.

Table 8.4 on page 141 shows PPL results on the validation set. The smoothed Fast NN LM approach improves the standard NN LM PPL in $\approx 7\,\%$ (36 absolut points), and the baseline SRI PPL in $\approx 27\,\%$ (180 absolut points).

Table 8.5 on page 142 shows WER results on the validation set, and Table 8.6 on page 142 summarizes the results on the test set. A statistical significant improvement is obtained by NN LMs, $\approx 9.2\,\%$ of WER (1.6 absolut points).

Baseline results analysis

The standard N-gram baseline shows a clear loss in performance with incresing values of N. The best result is obtained by bigrams. The size of the vocabulary provides a statistical significant improvement (comparing **mKN** with $\Theta = 20$K **mKN**) on CER and WER.

Input vocabulary size

This analysis will be focused on Fast NN LMs (**FNN**) models. Looking at PPL results (Table 8.4 on page 141), there exists a clear trent to worse results with bigger vocabularies. Nevertheless, this trent is not clear at WER results, where the differences are not significant. This result is totally task-dependent, and it is not applicable to other tasks.

Comparing NN and FNN approaches

Standard NN LMs (**NN**) and smoothed Fast NN LMs (**FNN**) show interesting and a priori surprising results. The **FNN** system is working better in all cases. However, a deeper understanding of this results leads us to the special characteristics of HTR *short* lines recognition. **FNN** approach has better accuracy at the beginning of the line recognition decoding, as Table 8.7 on page 144 shows.

Comparing N-best rescoring and integrated systems

Differences between the two approaches are not significant. Again, the recognition of short text lines leads on a very restricted search space, therefore, an N-best list is enough to represent this space.

Test set results analysis

Table 8.6 on page 142 shows a clear improvement using NN LMs (1.7 absolut points of WER). This difference is statistical significant for a $95\,\%$ confidence interval.

D.8.3. Character-based NN LMs experiments

See [Zamora-Martínez *et al.*, 2010].

D.8.4. Word-based experiments with recurrent neural network LMs

See [Frinken *et al.*, 2012].

D.8.5. Summary

This section has shown the experiments in HTR. A clear improvement of NN LMs over standard N-grams was found. The most important achievements are:

- Successfully compare different NN LM configurations versus standard N-gram LMs on HTR tasks. This experimental work is being prepared to the IEEE Transactions on Neural Networks journal [Zamora-Martínez *et al.*, 2012b].

- In collaboration with other authors, NN LMs have been evaluated with HTR systems based on recurrent BLSTM models. In this case the performance was also improved. Additionally, a ROVER combination of HMM/ANN and BLSTM systems achieves more impressive improvements. This work has been submitted to the Pattern Recognition journal [Zamora-Martínez *et al.*, 2012c].

- Research on the use of character-based NN LMs for HTR tasks, showing that it is possible to achieve an impressive improvement of the system using neural networks. This result opens a way to deal with old handwritten documents, where it is difficult to find enough material to train word-based LMs or, even, to determine the vocabulary of the task. This research was published in [Zamora-Martínez *et al.*, 2010].

- The use of recurrent neural networks, LSTM neural networks, seems promising. Neverhteless, longer sentence or paragraph recognition tasks will be needed to exploit all the capabilities of LSTMs. This research was published in [Frinken *et al.*, 2012].

D.9 Statistical Machine Translation

This section introduces the basic ideas behind Statistical Machine Translation (SMT) and the approach based on N-grams.

SMT could be formalized following the Equation (1.2) on page 8:

$$\hat{\bar{y}} = \arg\max_{\bar{y}\in\Omega^+} p(\bar{y}|\bar{x}) = \arg\max_{\bar{y}\in\Omega^+} p(\bar{x}|\bar{y})p(\bar{y}) = \quad (1.2)$$

$$= \arg\max_{\bar{y}\in\Omega^+} p(\bar{x},\bar{y})\,, \quad (9.1)$$

as it was stated before in HMM sequence recognition tasks. However, due to the discrete representation of the input space (input source language sentence $\bar{x}$) and the output space (output target language sentence $\bar{y}$), it is possible to compute the joint probability as it was shown at Equation (9.1).

SMT models are based on word alignments (see Figure 9.2 on page 164), therefore, first approaches were based on stochastic dictionaries for word-to-word translation [Brown *et al.*, 1990].

Word-based approaches were extended to phrase-based SMT, extracting from word alignments all possible phrase segments that are *consistent* [Och & Ney, 2003; Koehn *et al.*, 2003] (see Figure 9.3 on page 165). A phrase is consistent if all the words in a segment are only aligned with words that belongs to the same segment in the other language.

As it was stated before, under the Maximum Entropy framework, is possible to naturally extend SMT equations to fulfill a log-linear combination of several information sources (or features/models) [Och & Ney, 2002]:

$$\hat{\bar{y}} = \arg\max_{\bar{y}} \prod_{m=1}^{M} \mathcal{H}_m(\bar{x},\bar{y})^{\lambda_m}\,. \quad (1.3)$$

D.9.1. N-gram-based SMT: stochastic finite state machines

The approach that has inspired this work is based on N-gram-based SMT [Mariño *et al.*, 2006]. The major difference between the N-gram-based and the phrase-based approaches is the phrase/tuple extraction procedure. In the N-gram-based approach, bilingual tuples are extracted from a unique segmentation of every pair of sentences. The training of the bilingual translation model is done following the Grammar Inference and Aligments for Transducer Inference (GIATI) approach [Casacuberta & Vidal, 2004].

The source part of the corpora is reordered to follow the target word order [Banchs *et al.*, 2005; Sanchis & Casacuberta, 2006; Costa-jussà & Fonollosa, 2006]. This step ensures resuability of tuples and enhances the model estimation. This process is called *unfolding* (see Figure 9.4 on page 166). This work follows some of decoding characteristics explained in [Banchs *et al.*, 2005; Mariño *et al.*, 2006] to implement a source word reordering decoding.

Extracting tuples

A unique segmentation in tuples is extracted from a word alignment of the bilingual corpus following these steps [Mariño *et al.*, 2006] (see Figure 9.4 on page 166):

1. The source part of each sentence is reordered to take into account the target word order.
2. Tuples are extracted as the smallest bilingual unit that is possible to define so that no word inside a tuple is aligned with a word outside the same tuple.
3. The source part of each tuple is reordered in increasing order, that is, the words are monotonous inside the tuple, but there could be discontinuities.

This definition builds a segmentation of the reordered source sentence and the target sentence, where there could be tuples with zero or more words in the source or in the target parts. Zero words in the source part means that the search procedure will need to insert tuples. This behavior could be hard to implement in the decoder. Then, for simplicity, the target part of empty source tuples is added to the next or the previous tuple maximizing the probability of the lexicon direct and inverse models.

D.9.2. Modeling

To take into account the implemented constrained reordering search and the tuple-based approach, a generalization of the maximum entropy formula of Equation (1.3) on page 8 is:

$$(\hat{\bar{\mathcal{T}}}, \hat{\varphi}) = \arg\max_{(\bar{\mathcal{T}},\varphi)} \prod_{m=1}^{M} \mathcal{H}_m(\bar{\mathcal{T}}, \varphi)^{\lambda_m}, \qquad \boxed{9.3}$$

where $\bar{\mathcal{T}} = \mathcal{T}_1\mathcal{T}_2 \ldots \mathcal{T}_{|\bar{\mathcal{T}}|}$ is a sequence of target tuples $\mathcal{T}_i = (x_i, y_i) \in \Delta$, with $x_i \in \Sigma^+$ (one or more source words), and $y_i \in \Gamma^\star$ (zero or more target words). The vocabulary Δ is the bilingual vocabulary of the N-gram translation model. The function $\varphi : \{1, 2, \ldots, |\bar{x}|\} \rightarrow \{1, 2, \ldots, |\bar{\mathcal{T}}|\}$ associates a tuple index to each source word position of the sentence $\bar{x}$. The model restricts the order of source words inside a tuple to be always in an increasing order. The target sentence $\hat{\bar{y}}$ is extracted from the sequence $\hat{\bar{\mathcal{T}}}$ taking into account the target part y_i of each tuple.

N-gram translation model. The N-gram translation model [Mariño *et al.*, 2006] computes the approximation of $p(\bar{x}, \bar{y}) \approx p(\bar{\mathcal{T}})$ over the composition of the source and target sentences into a sequence of bilingual tuples, given the sequence of tuples. This model is a Stochastic Finite State Transducer trained from the bilingual tuple segmented corpus, following the GIATI technique [Casacuberta & Vidal, 2004]:

$$\mathcal{H}_{TM}(\bar{\mathcal{T}}, \varphi) = p(\bar{\mathcal{T}}) \approx p(\mathcal{T}_1 \mathcal{T}_2 \dots \mathcal{T}_{|\bar{\mathcal{T}}|}) \approx \prod_{i=1}^{|\bar{\mathcal{T}}|} p(\mathcal{T}_i | \mathcal{T}_{i-N+1} \dots \mathcal{T}_{i-1}) \,,$$

where N is the order of the N-gram translation model. In our system, this translation model is a NN LM, and will be denoted by NNTM.

N-gram target language model. The N-gram target language model computes the approximation of $p(\bar{y})$ as:

$$\mathcal{H}_{LM}(\bar{\mathcal{T}}, \varphi) = p(\bar{y}) \approx \prod_{i=1}^{|\bar{y}|} p(y_i | y_{i-N+1} \dots y_{i-1}) \,,$$

where N is the order of the N-gram target language model. Again, a NN LM is used as the target language model in our translation system, and will be denoted by NNTLM.

Other models. Besides the translation and the target LMs, the following models were added to the SMT system:

- Lexicon direct and inverse translation models: Both are based on IBM-1 word-alignment models [Brown *et al.*, 1993]. Let $q(y_j|x_i)$ the direct statistical dictionary probability for the alignment of source word x_i with target word y_j:

$$\mathcal{H}_{f2e}(\bar{\mathcal{T}}, \varphi) = \sum_{i=1}^{|\bar{\mathcal{T}}|} h'_{f2e}(\mathcal{T}_i) \,,$$

$$h'_{f2e}(x, y) = \frac{1}{(|x|+1)^{|y|}} \prod_{j=1}^{|y|} \sum_{i=0}^{|x|} q(y_j | x_i) \,,$$

  being the inverse direction models, $\mathcal{H}_{e2f}$ and h'_{e2f} computed in a complementary way.

- "Weak" distortion model: It computes a penalization of the reordering of the output hypothesis penalizing adjacent tuples which source part is not consecutive. The order of words inside a tuple is not penalized.

- Lexicalized reordering model: Six different models were trained, based on the Moses lexicalized reordering model [Koehn et al., 2007], and they were computed over the current and previous tuple, and the φ function.

- Word and tuple bonus models: These two models penalize the number of inserted words (WIP) and inserted tuples (TIP) generated by the system.

D.10 Decoding algorithm for SMT

The decoder has been implemented into the `April` toolkit, developed in our research team for pattern recognition tasks. Figure 10.1 on page 181 illustrates the `dataflow` diagram of the SMT decoding system. The search procedure was divided in three coupled steps:

1. Reordering module: The source word constrained reordering graph was generated in a topological order, following local constraints (a node is expanded covering every source word position where the distance between the first uncovered position and the last covered position is less than a certain given value). This approach is inspired by those presented in [Tillmann & Ney, 2003; Sanchis & Casacuberta, 2006; Costa-jussà & Fonollosa, 2006]. Our system scores the reordering hypothesis in the core of the SMT system. Histogram pruning can be applied during this step, using an approximation of the future cost of each reordering hypothesis based in Moses future cost computation.

2. Word2Tuple module: The previous graph is extended by forming tuples. Every sequence of source words is substituted with an edge between the first and last node of the sequence, tagged with all the possible tuples that were found given the source words. Source words order is restricted to be in increasing order inside a tuple. The graph is generated in a topological order. In this step the lexicon models, and word and tuple bonus models are added to the graph edges. This step is based in the GIATI technique [Casacuberta & Vidal, 2004].

3. Viterbi module: A Viterbi decoding with beam search and histogram pruning, using the N-gram target LM(s), N-gram bilingual translation LM(s), and the reordering models, is performed. This step outputs the 1-best, an N-best list, or a word graph following the Eppstein algorithm [Eppstein, 1998; Jiménez & Marzal, 2003]. GIATI technique allows to use the N-gram bilingual translation LM as a stochastic finite state transducer that generates the output sentence $\bar{y}$ given the best sequence of tuples $\bar{x}$.

The decoder could be extended with NN LMs during the Viterbi decoding, or in an N-best list rescoring decoupled stage.

D.10.1. Word reordering graph generation

This procedure is addressed by the first `dataflow` module. It implements a dynamic programming algorithm that solves the Travelling Salesman Problem (TSP) applied to SMT: it looks for a source word reordering that maximizes the probability of the translation. The enumeration of all possible word orders has a complexity of $O(2^{|\bar{y}|})$, being impossible a full search over this space.

In this work the search will be constrained following a histogram prunning scheme and some standard heuristics [Tillmann & Ney, 2003]. An approximate future cost of each word order hypothesis will be needed to ensure the performance of the histogram prunning.

The algorithm has three parameters:

- histogram prunning: it will limit the number of expanded hypothesis in each dynamic programming stage;

- initial beam search: it will indicate the size of the beam search at the first stage of the dynamic programming algorithm;
- final beam search: it will indicate the size of the beam search at the last stage of the dynamic programming algorithm. Intermediate stages will have an interpolation of initial and final beam search value. Note that the dynamic programming graph will have $|\bar{x}|$ stages, one for each source word.

This procedure is described by Algorithm 10.1 on page 186. This algorithm uses the best probability found by the rest of the SMT system at each word-graph node. The line 31 asks the next `dataflow` module to return the best probabilities on current stage word-graph nodes.

The algorithm follows these heuristics to constrain the search space [Berger *et al.*, 1996b; Matusov *et al.*, 2005]:

- IBM-l constraint: it delays the translation of few source words, covering always the first l uncovered words.
- IBM-inverse-l constraint: it was developed for some language pairs that needs to translate a little portion of the sentence ending and then continues with an almost monotonous decoding. Being j the first uncovered position, this constraint is implemented choosing any uncovered position j' until $l-1$ positions $j' > j$ were translated. Otherwise, it will translate the position j.
- Local constraint with window l: it was developed for languages where only a few words need to be reorderd a few positions to the left or right, being the major part of the sentence monotonous decoded. The constraint choose uncovered positions in a window of size l surrounding the first uncovered position.

Figure 10.4 on page 188 shows instances of the reordering graph generated following this constraints.

The future cost of each reordering graph node is computed following the Moses approach [Koehn et al., 2007], with an additional simplification to the bilingual translation model future cost (only the unigram probability will be used). Algorithm 10.2 on page 190 describes the followed procedure to compute a matrix with all possible future cost computations given a sequence of source words $\bar{x}$.

D.10.2. Tuple graph generation: Word2Tuple

This module expands with tuples (or phrases) all possible paths in the word reordering graph. It applies a transitive clausure to the input graph. Figure 10.5 on page 192 is an instance of the word reordering graph where some tuple extensions where added in gray color. Figure 10.6 on page 192 is a path in previous graph. Following the graph protocol, the implemented algorithm works running some special methods of the graph protocol:

- `vertex` (id): it indicates that a new active vertex with id identifier was generated (see Algorithm 10.3 on page 194).
- `is_initial` (p): it makes the active vertex an initial vertex with probability p.
- `change_stage`: this signal indicates that a stage change was produced, therefore, all the next vertices will be associated with the new stage.

- `edge` (id, data=$\{\langle r, p\rangle\}$): it indicates an edge between the vertex id and the current active vertex, using the source word x_r (at position r), with probability p, normally set to $p = 1$ (see Algorithm 10.4 on page 195).
- `no_more_in_edges` (id): this message closes the vertex id, it will never received new incoming edges (see Algorithm 10.5 on page 196).

The resulting tuple graph will contain, at the edges, the probability of the log-linear combination of the lexical models, the direct and inverse conditional translation probabilities, and the hypothesis size penalties.

D.10.3. Best tuple path search: Viterbi module

This module implements an adapted version of the Viterbi algorithm to work with graphs. This module uses the edge probabilities computed previously, and applies the LMs (target language model and bilingual translation model) and reordering models probabilities. This module also implements methods of the graph protocol, being the most important the `edge` method (see Algorithm 10.6 on page 198).

At the end of the translation, this module can generate the 1-best path, an N-best list, or a graph with the N-best hypotheses.

D.10.4. Execution example

Figure 10.7 on page 200 illustrates an execution with the input instance "tengo un coche rojo". Figure 10.8 on page 201 is a detail of the previous execution, focusing in the Viterbi module. The figure shows all possible active states, grouped by input graph node. Bolded lines are backpointers of the 1-best path. All LMs are bigrams. Each active state shows:

- TLM LMkey: identifier of the target LM state.
- BLM LMkey: identifier of the bilingual translation LM.
- Last tuple: incoming transition tuple.
- Seg. info.: information about the initial and final covered positions in the source sentence.
- Backpointer: backpointer to previous active state.

D.10.5. Summary

This section has described the algorithms that implements the SMT system. Its goals were to show:

- the generality of the sequence recognition developed algorithms, which were minimal adapted to work in SMT tasks;
- the integration of NN LMs into the decoding stage.

All algorithms are described in a reactive way, describing what to do in the graph protocol messages.

D.11 Statistical Machine Translation experiments

D.11.1. Experiments with N-best list rescoring

Some works were done to evaluate the developed NN LMs in an N-best list rescoring scheme. The experiments could be classified in:

- Two preliminar works, Italian-English IWSLT'06 task published in [Khalilov *et al.*, 2008b], and Arab-English NIST task published in [Khalilov *et al.*, 2008a], using the N-gram-based decoder Marie [Crego *et al.*, 2005b,a]. A paper is being prepared to extend these two preliminar works with new results [Khalilov *et al.*, 2012].

- Two participations in the Workshop Of Machine Translation (WMT), in 2010 and 2011 [Zamora-Martínez & Sanchis, 2010; Zamora-Martinez & Castro-Bleda, 2011]. These two works rely only on English-Spanish translation direction using the toolkit Moses [Koehn et al., 2007]. Tables C.1 and C.3 in appendix C show the official ranking in both evaluation champaings.

Readers are referred to the published papers where all details of experiments can be found.

D.11.2. Experiments with NN LMs coupled at the decoding phase

Two works were presented in this section:

- A participation on the French-English BTEC task at the International Workshop of Spoken Language Translation (IWSLT) 2010 edition [Zamora-Martinez *et al.*, 2010]. Table C.2 shows official results on this campaign.

- Experimentation with News-Commentary 2010 corpora, extracted from WMT'10 champaign.

A summary of these two experiments was submitted to the COLING'12 conference [Zamora-Martinez & Castro-Bleda, 2012]; therefore, details can be found in this paper.

D.11.3. Summary

This section summarizes the experimental results in SMT using NN LMs. Reference system performance improvements were achieved in all conducted experiments. BLEU improvements were between 0.5 absolut points and 2.4 absolut points, depending on the task, achieving statistical significant improvements under a pairwise comparison test for a confidence interval of $95\,\%$.

The system that integrates NN LMs into the decoding achieves better performance using the N-gram-based approach. The final system is two or three times slower than the standard N-gram-based system, so more efforts are needed to reduce the computational cost.

This section has achieved these scientific goals:

- Validation of NN LMs in SMT tasks via a decoupled N-best list rescoring. Published in [Khalilov *et al.*, 2008b], [Khalilov *et al.*, 2008a], [Zamora-Martínez & Sanchis, 2010], [Zamora-Martinez & Castro-Bleda, 2011].

- A phrase-based translation model tables combination scheme was proposed, obtaining improvements of 0.5 BLEU points. Published in [Zamora-Martinez & Castro-Bleda, 2011].
- Competitive performance was achieved using an N-gram-based system that integrates NN LMs into the decoding stage. Published in [Zamora-Martinez *et al.*, 2010].
- An extension of the previous system using a larger task was proposed. A classification of tuples in statistical classes was needed in order to make the NN LM training feasible (for the bilingual N-gram translation model). This research was submitted to the COLING 2012 conference [Zamora-Martinez & Castro-Bleda, 2012].

D.12 Conclusions and future work

The conclusions are written in English in Chapter 12 on page 235.