UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

Language modeling for streaming automatic speech recognition

End of Degree Project

Bachelor's Degree in Informatics Engineering

AUTHOR: Mas Mollà, Gerard

Tutor: Sanchis Navarro, José Alberto

Cotutor: Silvestre Cerdà, Joan Albert

Experimental director: GARCES DIAZ-MUNIO, GONZALO VICENTE

ACADEMIC YEAR: 2021/2022

Resum

El reconeixement automàtic de la parla (ASR per les seues sigles en anglès) és una branca del reconeixement de patrons que ha vist augmentada la seua popularitat degut als grans avanços tecnològics i estructurals d'aquest camp. Aquest treball presenta el procés de creació d'un model de llenguatge en francès que formarà part d'un sistema de reconeiximent automàtic de la parla francesa. El procès de creació del model de llenguatge es descriu partint des de l'inici, examinant i descrivint també els conjunts de dades emprats per entrenar el sistema. Finalment, el sistema final, un sistema d'ASR híbrid comformat per el model de llenguatge d'aquest treball juntament amb un model acústic prèviament desenvolupat per investigadors del grup *MLLP-VRAIN*, es compara amb una versió anterior d'un sistema ASR hibrid en francès del mateix grup.

Paraules clau: reconeixement de formes, aprenentatge automàtic, reconeixement automàtic de la parla, aprenentatge profund, modelat de llenguatge, streaming, francès

Resumen

El reconocimiento automático del habla (ASR por sus siglas en inglés) es una rama del reconocimiento de patrones que ha aumentado su popularidad debido a los grandes avances tecnológicos y estructurales de este campo. Este trabajo presenta el proceso de creación de un modelo de lenguage en francés que formará parte de un sistema de reconocimiento automático del habla francesa. El proceso de creación del modelo de lenguage se describe partiendo desde el inicio, examinando y describiendo también los conjuntos de datos utilizados para entrenar el sistema. Finalmente, el sistema final, un sistema ASR híbrido formado por el modelo de lenguage desarrollado en este trabajo y un modelo acústico préviamente desarrollado por investigadores del grupo *MLLP-VRAIN*, se compara con una versión anterior de un sistema ASR híbrido en francés del mismo grupo.

Palabras clave: reconocimiento de formas, aprendizaje automático, reconocimiento automático del habla, aprendizaje profundo, modelado de lenguage, streaming, francés

Abstract

Automatic speech recognition (ASR) is a branch of pattern recognition that has currently seen its popularity increased due to technological and structural advances in this field. This work presents the process of creating a language model in French that will be part of a French ASR system. The process of creating the model from scratch is described, also examining and describing the datasets used to train the system. To conclude, the performance of the developed system, which is a hybrid ASR system consisting on the language model developed in this work and an acoustic model previously developed by researchers from the *MLLP-VRAIN* group, is compared with a previous version of a French hybrid ASR system from the group.

Key words: pattern recognition, machine learning, automatic speech recognition, deep learning, language modeling, streaming, French

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

1.1 Motivation

Natural Language Processing (NLP) technologies aim to understand and dominate the main communication tool for humanity: language. Understanding and processing another person's voice is not an easy task but, as humans, we are able to do it due to years and years of training our ears and brains to do so. The field of automatic speech recognition (ASR) aims to give to the computers the ability of recognizing the spoken language, ability that remained reserved for humans up until the last century.

The French language is currently spoken by over 275 million people around the world, which places French in a similar level as English or Spanish when refering to relevance. This position of a super-spoken language caused the French to be an official language for many international organizations such as the NATO, the International Court of Justice or Amnesty International. This is also the case of the Conseil Européen pour la Recherche Nucléaire [1], better known as CERN, which has both English and French as its official languages.

This project is developed under the framework of a research and development (R&D) project relating the *Machine Learning and Language Processing (MLLP)* research group [2] of the *Valencian Research Institute on Artificial Intelligence* [3] *(VRAIN)* with the CERN, which aims to provide this institution with NLP tools to transcribe and translate, in English and French, all kinds of multimedia resources (meetings, conferences, courses...). To be more specific, this work aims to develop state-of-the-art language models (LM) to be integrated into an upgraded French ASR system, that will be deployed on CERN premises.

Aside from that, this work also aims to produce an streaming-ready ASR system, able to transcribe French speech in an streaming environment with minimal quality loss. At the end of the work, the resulting system will be compared in an offline environment to the previous French ASR system developed by the *MLLP-VRAIN* research group on June 2017, in order to assess the contribution of the technological upgrades considered in this work to the overall ASR system quality.

1.2 Main goals

The main goals of this work are the following:

[1] https://www.cern.ch
[2] https://mllp.upv.es
[3] https://vrain.upv.es

- To gather and apply all the concepts studied during this degree alongside with state-of-the-art techniques involved in training ASR systems for real-life applications.
- To interpret the underlying theoretical concepts of the ASR systems.
- To use state-of-the-art software and tools to build complex and competitive streaming-ready ASR systems.
- To evaluate the performance of ASR systems.
- To study the contribution of state-of-the-art language modelling techniques to the overall ASR system performance, by comparing our proposed ASR French systems with a former one.

1.3 Document structure

This document is divided into six chapters. This first chapter has exposed the motivation of this project, discussed general ASR concepts and explained the structure of this work. Next, Chapter 2 contains an introduction the ASR fundamentals, starting from the most basic pattern recognition application to the state-of-the-art when speaking about ASR. Then, Chapter 3 describes the procedure of training state-of-the-art language models for streaming ASR. Next, Chapter 4 presents the corpora involved in both the training or evaluation of the models. Then, Chapter 5 describes in detail the development of the LMs and their corresponding ASR systems, as well as their experimental results. Finally, Chapter 6 summarizes the work done, gives some concluding remarks, and outlines possible futures lines of work and investigation.

The reader is recommended to read the document sequentially, since each chapter introduces and explains concepts that will be used in the following ones. However, if the reader is experienced in this field, there are chapters that may be omitted. It is the case of chapter 2 and 3, if the reader is already familiar with ASR technology.

CHAPTER 2
Automatic Speech Recognition

This chapter gives an introduction into Automatic Speech Recognition (ASR), going through the basics of the technologies that underpin it. It is structured as follows. First, Section 2.1 introduces the concept of ASR, contextualizing it and giving the motivation to study this field. Second, Section 2.2 discusses the basics of pattern recognition. Third, Section 2.3 gives an overall view on the machine learning branch, while exposing some classical classifiers. Fourth, Section 2.4 goes through the basic concepts of Neural Networks (NN), from the perceptron linear classifier to the Transformer architecture. Then, Section 2.5 explains the goals of ASR, while exposing the hybrid approach to ASR and the two main components involved on it. Finally, Section 2.6 exposes the evaluation measures used to evaluate LMs and their counterpart ASR systems.

2.1 Introduction

ASR is a branch of pattern recognition that can be described as a process that, given an input speech represented as an audio signal, extracts the text corresponding to its transcription. It is also a very cheap and efficient process in comparison with human transcription, since the only requirements for starting to transcribe indefinitely are a computer and a trained model.

This branch has experienced an exponential growth in popularity due to recent technological advances, resulting in an increase of the computational power available and, therefore, the capability of building more powerful and accurate models. If we compare the performance of past ASR systems to the ones that are considered state-of-the-art nowadays, we can observe significant differences in error rate. For instance, considering a 2-year lapse, *Machine Learning and Language Processing - Universitat Politècnica de València* (MLLP-VRAIN-UPV) research group's 2018 best performing Spanish streaming ASR system developed for the Albayzín-RTVE 2018 Speech-To-Text Challenge [34] provided a score of 23.1 Word Error Rate (WER, see Section 2.6), whereas the system developed for the 2020 edition of the same challenge [32] scored 16.0 WER (28% relative WER improvement).

2.2 Pattern recognition

Pattern recognition is a research field that provides a widely used approach to recognize and classify samples using patterns or some criteria that allow to separate them in classes [22]. It comprises ASR, as well as other branches like computer vision, machine translation and Text-To-Speech.

The main goal of a pattern recognition system is to assign a class label c to a given sample. But in order to be able to correctly recognize samples, we first need to preprocess them, this is, extracting some "representative" features of each sample and storing them in a "feature vector" $\mathbf{x}$. Then, we need a classification function that maximizes the probability $p(c|\mathbf{x})$ of an object $\mathbf{x}$ belonging to a class c:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, p(c|\mathbf{x}) \tag{2.1}$$

Note that $\hat{c}$ refers to the estimated class that yields the maximum value of $p(c|\mathbf{x})$.

In some cases, specially when dealing with linear classifiers, the classification function is associated to Bayes' theorem, that allows to estimate $p(c|\mathbf{x})$ precisely. The Bayes' theorem relates the probability of an object $\mathbf{x}$ belonging to a class c to the probability of c yielding $\mathbf{x}$. With this idea in mind and applying this theorem, the classification function can be expressed as:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, \frac{p(\mathbf{x}|c)p(c)}{p(\mathbf{x})} \tag{2.2}$$

Since $p(\mathbf{x})$ is constant to c (it only depends on $\mathbf{x}$ itself), it can be safely removed from the denominator, since the aim of the equation is to minimize the classification error and, therefore, its success does not depend on the magnitude of the output values, but on the ratio between them.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, p(\mathbf{x}|c)p(c) \tag{2.3}$$

The statistical models used are the result of adjusting their parameters trying to minimize the error rate of classifying a given training set of samples. This process of "training" can either be supervised, if the training samples are labeled (we know $c_{\mathbf{x}}$ for every $\mathbf{x}$ in the training set), or unsupervised if they are not. In both cases, the goal is to classify unlabeled samples as accurately as possible.

Figure 2.1 shows a schematic representation of a generic pattern recognition system. Note that the process of data preprocessing and feature extraction is exactly the same for both training and recognition data, although the training data is the only one labeled. Here, "classification" and "recognition" are treated as synonyms and refer to the process of using the trained model for classifying the recognition data.

2.3 Machine learning

Machine learning is a branch of pattern recognition that focuses on data-driven models, aiming to build computer algorithms that are able to autonomously learn how to solve a task [44]. It is closely related to computational statistics, since these algorithms are usually implemented to estimate probabilities (like in the case of ASR).

Following the statistical application of machine learning, there are many typical binary classifiers used to classify linear data. Some of the simplest ones can be probabilistic distributions, where the learning algorithm makes use of the data to tune the distribution parameters (i.e. for a Gaussian distribution, the parameters to learn can be the mean μ and the squared variance σ^2) [27]. An extra iteration on these classifiers are mixtures of

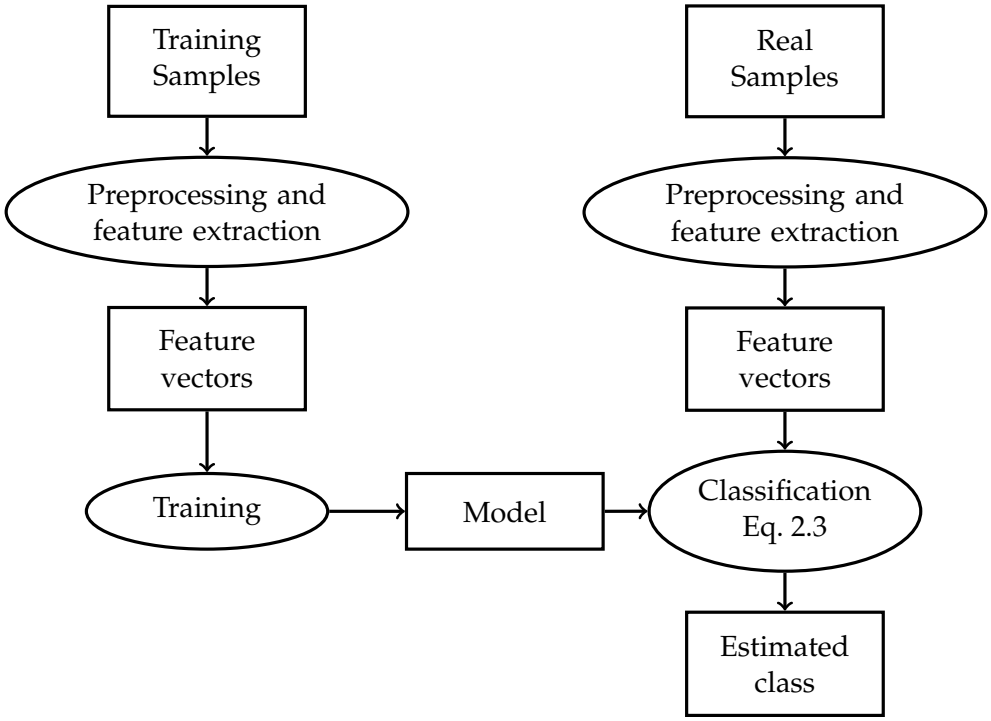


Figure 2.1: Generic pattern recognition system.

distributions, where k different distributions are trained for each class (giving the learning algorithm an extra parameter to learn) [35, 25, 63, 41].

Other more complex classifiers can be support vector machines (SVM) [18, 46], which are able to efficiently perform non-linear classification using spatial transformations, or neural networks, used in all kind of applications due to their powerful feature extraction capabilities.

2.4 Neural networks

It is a fact that the first concept that comes to mind when talking about machine learning are neural networks, especially after the exponential growth in popularity of this kind of models in the recent years. The introduction of neural models to ASR tasks, especially since the advent of RNNs, led to the emergence of neural-based acoustic and language models [12, 43, 30, 54].

However, to understand the concept of neural network and the usage of these models in this work, it is necessary to first understand the concept of the perceptron.

2.4.1. Perceptron and multi-layer perceptron

The perceptron [52] is an algorithm for supervised learning of linear classifiers. It is capable of determine whether or not an input corresponds to a specific class by combining a set of weights with the feature vector. It is also referred to as a function that maps an input vector $\mathbf{x}$ to a binary output value $f(x)$ that can be expressed as:

$$f(x) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

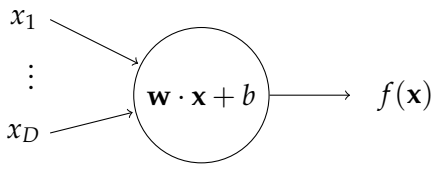


Figure 2.2: Representation of a perceptron as a neuron.

Where $\mathbf{w}$ is a vector of weights, $\mathbf{x}$ is the input feature vector and b is a bias parameter.

However, the conceptual representation of this model is often related with natural networks: it can be seen as a cell that receives an input vector through its dendrites and outputs the result of internally performing an operation over it through its axon terminals. Figure 2.2 shows the neuron representation of a perceptron for D dimensional vector classification.

This model gained in popularity as the computational power available increased, and it started to evolve by combining multiple perceptrons between them in connected layers as if they were neurons. The resulting combination is called multilayer perceptron [28] and it is the basis of today's modern neural network models. Figure 2.3 shows a multilayer percepton schema.

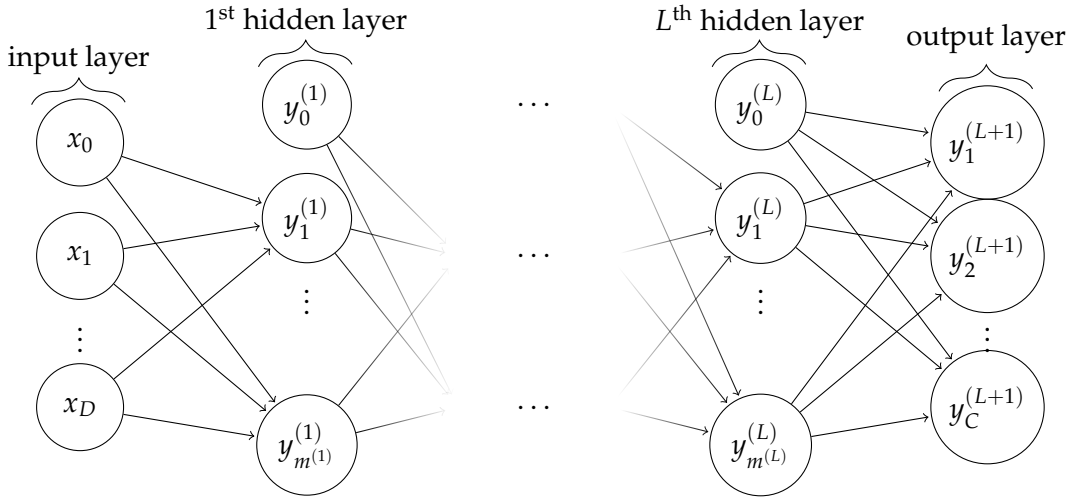


Figure 2.3: Representation of a multilayer perceptron. Layers are connected between them, being the input of the n-th layer the output of the $n-1$-th.

However, since the perceptron is a linear classifier, the multilayer perceptron is still limited to classify linear separable data. To overcome this problem, the concept of activation gate or function was introduced.

An activation gate is a non-linear function used to transform the linear information of a neuron to non-linear. They are appended to the output of each neuron in order to grant the neural network the capability of dealing with non-linear data. Examples of activation functions are ReLU, Sigmoid or Softmax [39]. These functions allowed the usage of error back-propagation algorithms [53] and opened up the possibility of training bigger and more complex models.

The multilayer perceptron is usually used for both classification and regression. In classification, the n-th node of the output layer is in charge of yielding the probability of the input vector belonging to the n-th class (for example, a computer vision model). In this kind of applications, a softmax gate is applied to the output layer. The softmax function is used to yield a probability distribution considering every class (node), since

it is bounded between 0 and 1 and it is capable of computing a unary probability mass. It is expressed as:

$$\sigma_M(z_n) = \frac{exp(z_n)}{\sum_{m=0}^{N} exp(z_m)} \tag{2.5}$$

Being n the identifier of the selected output node and N the size of the output layer.

At the same time, in regression [37], the output layer yields a vector that can be used for representing an object related to the input vector (for example, outputting a picture based on a text description [50]).

As said, these models had a clear tendency to grow in size and complexity, a fact that was supported by the growing computational capacity available. This resulted in the introduction of deep neural networks (DNNs) [38] to analyze increasingly complex data.

In this context, the multilayer perceptron concept started to fade out, since the different DNN models started to be refered to by their functionalities. In the case of DNN classifiers (multilayer perceptrons with non-linear activation and a softmax output layer), the concept of feed forward DNN (FF-DDN) [10] became more suitable. In these models, the information is only propagated in one direction, hence the name.

2.4.2. Recurrent neural networks

Recurrent neural networks (RNNs) [55] are a type of neural networks specifically designed to cope with sequential data, where the input information present in the i-th timestep of the sequence might depend on that from previous and/or future timesteps. This is the case of text (sequence of words), handwritten text (sequence of typestrokes) or speech (sequence of uttered phonemes), among others.

This type of models take FF-DNNs as basis and, for every layer, its output is also connected to its input. By doing this, RNNs are able to re-use information that was already seen in the past. In this kind of models, it is convenient to use the term "cell" when referring to a unit of the model.

Figure 2.4 shows the schematic concept of RNNs.The output vector of the net is reused appending it to the next input vector, allowing the RNN to consider the context of sequential data. The appended part of the input vector is set to a specific value in the first iteration.

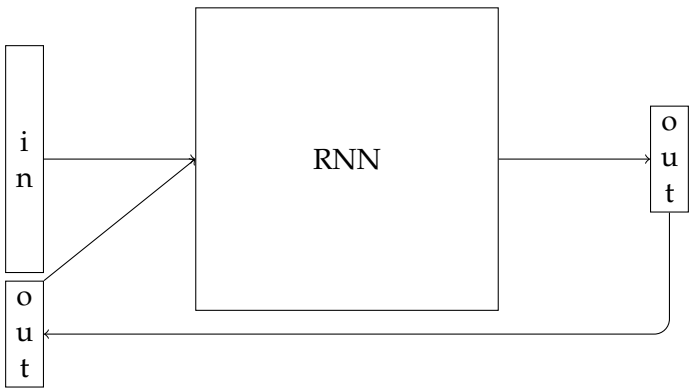


Figure 2.4: Representation on how the mechanism of an RNN cell works.

This property made them very popular in ASR tasks (in both acoustic [54] and language [43] modeling), Neural Machine Translation or other simpler linear tasks such as

learning how to count, as FF-DNNs had not yet been successfully applied in a standardized way to this kind of problems (or at least not with good results). However, since the memory information is updated in every iteration of the net, it tends to fade out, resulting in lack of long-term memory. This is the so-called gradient vanishing problem [48]. To solve this problem, the LSTM networks emerged [55].

2.4.3. Long-short term memory models

Essentially, Long-Short Term memory (LSTM) [31] are RNNs with a more complex memory mechanism where the information is updated depending on the input. They feature a three-step memory control mechanism with three different gates apart from the standard input. Figure 2.5 shows a basic schema of LSTM unit.

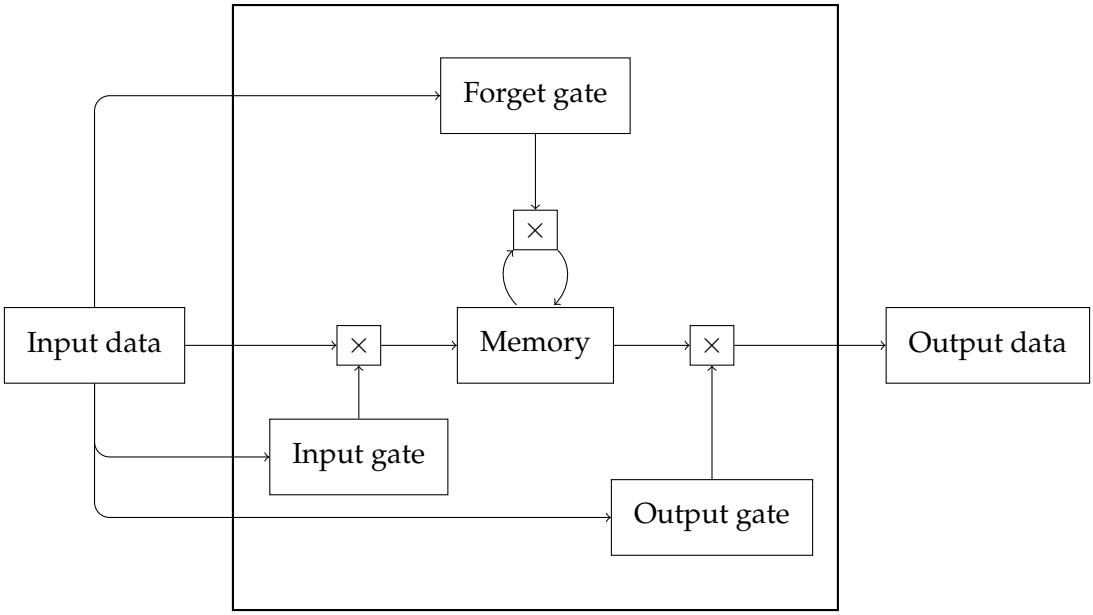


Figure 2.5: Schema of the memory mechanism of an LSTM cell. Nodes labeled with $\times$ denote the multiplication of their inputs.

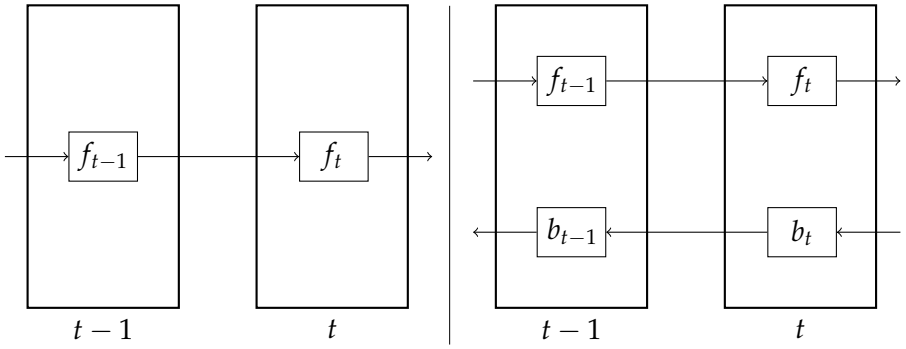
First, the Input gate is applied. It receives the input vector and returns a value in the range $[0, 1]$ that is multiplied to the input vector before entering the memory cell. It acts as a throttling factor that decides how important it is to consider the usage of the input. Once in the memory cell, the Forget gate is applied. It works in the same way as the Input gate, this time designating the importance of remembering the input value (and therefore forgetting the present contents of the cell, hence the name). Finally, the Output gate is applied. This gate, again, returns a throttling value based on the input that controls the importance of propagating the cell information to the following units.

Using these control gates, the LSTM cell works in a similar way to a computer memory cell, allowing the model to remember potentially infinite histories.

However, the shortcoming of the LSTM nets is the sequential nature of the model, being able to analyse data dependencies only in one direction. This led to a limitation in results for tasks such as ASR, as words tend to make both forward and backward references to other words.

To overcome this limitation, LSTM networks evolved to analyze data in both directions, giving rise to the Bidirectional LSTM (BLSTM) [51] model. The latter can be described as two LSTM networks working at the same time, one in each direction.

Figure 2.6 shows the idea of BLSTM bidirectionality where each frame represents a time instant t, with nodes f_i and b_i being the states of the nets in such instant t. Nodes f_i are responsible for the forward processing of the input data, while b_i nodes are in charge of processing it backwards.



(a) Long short term memory network.

(b) Bidirectional long-short term memory network.

Figure 2.6: Long-short term memory network (left) vs bidirectional long-short term memory network (right).

BLSTM networks represented an important improvement in ASR, since bidirectionality allowed to detect references to words in the future. Still, in practice, having to process the input data twice (once per direction) is a time-expensive task which represents a latency problem for streaming applications. In addition, it was proven that although LSTMs were supposed to overcome the problem of gradient vanishing, these models still struggled to capture very long-term dependencies.

In order to solve it, the focus of investigation shifted towards attention-based models [17]. These models included an attention mechanism that iteratively processes the input and selects the most important content for each step.

When combined witch recurrent models, the attention mechanism processes the input in both directions and obtains an attention vector with the encoded history for each word. However, the temporal complexity for these mechanisms is $O(n)$ ($O(logn)$ in the most efficient systems), where n the size of the input. This, combined with the complexity of BLSTM networks, resulted in huge models reserved almost exclusively for offline applications, where they could unfold their full potential.

In this context, in order to avoid processing the input data sequentially and seeking the capability of capturing very long-term dependencies, the transformer architecture was introduced.

2.4.4. Transfomer architecture

The Transformer architecture [64] relies on the so called self-attention mechanism to compute dependencies in a sequence. In essence, self-attention is an attention mechanism that relates different positions of a single sequence in order to assign them an attention weight. In this way, self-attention is able to compute the importance of relating two elements of a single sequence, in a similar way to how a syntactic analysis is done.

This architecture features an encoder-decoder structure. Here, the encoder maps an input sequence of symbols x to a sequence of continuous representations z and uses it as input. Given z, the decoder is in charge of mapping again the result to a sequence of output symbols y. In addition, both the encoder and decoder feature stacked self-attention and Feed Forward layers.

Aside from the simplicity of self-attention, the Transformer instantly positioned itself as the go-to model when developing state-of-the-art systems to analyze sequences, due to the reduced time complexity of the self-attention mechanism, aside of being able to handle potentially infinite histories. When comparing to the previously used attention mechanisms with $O(n)$ time complexity, self-attention only requires $O(1)$ sequential operations, since it is able to compute the attention vector for every word in the sequence in parallel.

Following these technical advances, the Transformer was also quickly introduced to the ASR field, usually outperforming the well-established recurrent models [21, 66, 9].

2.5 Automatic speech recognition

ASR aims to obtain a sequence of words $\hat{w}$ that most likely correspond to the speech utterances present in an audio stream $\mathbf{x}$ given as an input. Giving it a pattern recognition approach, the sample is the sequence of acoustic vectors $\mathbf{x}$ given as input and the classes are every single word w in vocabulary of the considered language L. Formalizing the problem, we need to obtain the sequence of words $w = w_1 w_2 ... w_N$ that maximizes the probability $p(w|\mathbf{x})$ given the sequence of acoustic feature vectors $\mathbf{x} = x_1 x_2 ... x_T$.

On the one hand, the probability of $p(w|\mathbf{x})$ can be directly estimated using a sequence-to-sequence (seq-to-seq) model [16], where given an input sequence $\mathbf{x}$, a decoded sequence w is yielded. These models aim to simplify the process of complex tasks such as ASR or Machine Translation by using only one model. However, this approach requires significant amounts of labelled (transcribed) speech data for training the underlying model, whose availability, in some cases, can be particularly scarce.

On the other hand, applying the Bayes' theorem, the probability $p(w|\mathbf{x})$ can be simplified to:

$$\hat{w} = \underset{w \in L^*}{\operatorname{argmax}}\, p(\mathbf{x}|w)p(w) \tag{2.6}$$

Where L^* represents the set of all possible sequences of words in the language L, $p(w)$ is the probability of w being an admissible sequence of words in the target language L and $p(\mathbf{x}|w)$ is the probability of w generating the acoustic sequence $\mathbf{x}$. This equation corresponds with the hybrid approach to ASR.

In a hybrid ASR system, two models take place. The model in charge of estimating $P(\mathbf{x}|w)$ is the acoustic model (AM). On the other side, the language model (LM) estimates the probability of $P(w)$. This approach allows to exploit more abundant sources of data, since the LM only requires monolingual text data in order to be trained. Hence, the hybrid approach allows to build robust, powerful LMs that can significantly expedite raw ASR performance. However, since each of the components is of a different nature, they work with different probabilistic scales of magnitude. For this reason, to avoid this difference in magnitudes causing the LM contribution to become irrelevant for the decoder, a scaling factor can be applied to Equation 2.6 in order to scale the probability yielded by the LM. Hence, the equation of the hybrid approach to ASR would be written as follows:

$$\hat{w} = \underset{w \in L^*}{\operatorname{argmax}} \log p(\mathbf{x}|w) + \alpha \log p(w) \tag{2.7}$$

Being α the scaling parameter for the LM probability. This parameter needs to be empirically explored in the decoder testing step, being its optimal value the one that

minimizes the WER w.r.t. a certain development set of data. Using this factor, the probability yielded by the LM is scaled in order to guide the decoder to output more reasonable answers. Indeed, this work focuses language modeling for the hybrid ASR approach.

2.5.1. Language model

As said in the previous section, the language model (LM) is the module in charge of computing $p(w)$. For this, it aims to represent the structure of the language and gives the probability that a sequence of words w is valid for a language L. It typically predicts the probability of a word w_i appearing right after a context of previous words $w_j...w_{i-1}$ (history) as $p(w_i|w_j...w_{i-1})$ where $j < i$. All probabilities are subject to:

$$\sum_{w \in L^*} P(w) = 1 \tag{2.8}$$

Many techniques have historically been used to model P(w). Formerly, count-based n-gram models were used, but, more recently, in the advent of neural networks, RNNs [43], LSTM [15] and Transformer [66] models have been vastly explored, reporting significant performance improvements over n-grams. In this work, we explore language modelling with n-gram models and Transformers.

Count-based n-gram models

One of the most used approaches to the language modeling are n-gram models. In the ASR context, n-grams are contiguous sequences of n words (for example, the sequence *"automatic speech recognition"* could be represented as the trigram [*automatic, speech, recognition*]), and are usually referred by their length (unigram, bigram, trigram, four-gram...). With this in mind, these model use n-grams to approximate $p(w)$ to the probability of a word w_i given a history of $n-1$ words (the sequence $w_{i-n+1}...w_{i-1}$) or, expressed in probability terms, $P(w_i|w_{i-n+1}...w_{i-1})$. This can be formalized as:

$$p(w) \approx \prod_{i=1}^{I+1} p(w_i|w_{max(i-n+1,0)}^{i-1}) \tag{2.9}$$

Where I is the length of w, n is the size of the n-gram, w_i^{i+j} represents the sequence of words $(w_i, w_{i+1}, ...w_j) \in w$ and w_0 and w_{I+1} represent the special first and last words respectively. Following this idea, the probability for "good morning" with trigrams (3-grams, $n = 3$) would be expressed in the following way:

$$\begin{aligned} p(\text{"good morning"}) &= p(\text{"good"}|START) \\ &= p(\text{"morning"}|START, \text{"good"}) \\ &= p(END|\text{"good"}, \text{"morning"}) \end{aligned} \tag{2.10}$$

Building an n-gram model is relatively easy, since the probabilities for a given n-gram can be empirically estimated by counting the number of occurrences of that n-gram in the training corpus C and normalizing to the number of occurrences of the lower order n-gram represented by the history [24]. This can be formalized as:

$$p(w_i|w_{i-2}w_{i-1}) = \frac{N(w_{i-2}w_{i-1}w_i)}{N(w_{i-2}w_{i-1})} \tag{2.11}$$

Where $N(w_{i-2}w_{i-1}w_i)$ is the number of occurrences of the trigram $w_{i-2}w_{i-1}w_i$ in the training set C, whilst $N(w_{i-2}w_{i-1})$ is the number of occurrences of the history (bigram) $w_{i-2}w_{i-1}$. As shown in equation 2.8, the n-gram model needs to estimate a probability distribution, so it can be said that:

$$N(w_{i-2}w_{i-1}) = \sum_{w_i \in C} N(w_{i-2}w_{i-1}w_i) \tag{2.12}$$

This approach is useful for computing probabilities to sentences seen in the training set. However, it is unable to assign a non-zero probability to a sentence that was not seen in the past. To overcome this problem, there are smoothing techniques that can be applied to n-gram models.

Let's say the word sequence "bees love singing" is presented to the n-gram model, which has an apparently low probability. If the word "singing" does not appear after the history "bees love" in the training set, instead of giving it probability zero, the model can check if "singing" appears after "love" and if "singing" is a common word in the training set. Following this idea, the equation 2.11 can be rewritten as:

$$p(w_i|w_{i-2}w_{i-1}) = \lambda_1 \cdot \frac{N(w_{i-2}w_{i-1}w_i)}{N(w_{i-2}w_{i-1})} + \lambda_2 \cdot \frac{N(w_{i-1}w_i)}{N(w_{i-1})} + \lambda_3 \cdot \frac{N(w_i)}{N} + \lambda_4 \tag{2.13}$$

Where $0 \leq \lambda_i \leq 1$, $\sum_i \lambda_i = 1$, $\forall i \in [1,4]$ and N is the total number of words in the training set.

This smoothing technique is called lineal interpolation, since it defines a probability distribution out of a linear combination of the distributions from other models (in this case, unigrams, bigrams and trigrams). This way, the probability for an n-gram is equal to a weighted distribution from the already trained models, as well as a small fixed probability mass (λ_4). The weight parameters λ_i are usually tuned aiming to optimize a development set. Considering the previous example, the probability of the sequence "bees love singing" can be computed as:

$$\begin{aligned} p(\text{"singing"}|\text{"bees love"}) = \lambda_1 &\cdot \frac{N(\text{"bees love singing"})}{N(\text{"bees love"})} \\ + \lambda_2 &\cdot \frac{N(\text{"love singing"})}{N(\text{"love"})} \\ + \lambda_3 &\cdot \frac{N(\text{"singing"})}{N} \\ + \lambda_4 & \end{aligned} \tag{2.14}$$

For further knowledge in n-gram estimation and smoothing, the reader is pointed to [40].

Neural language models

Leaving the statistical approach for language modeling, neural language models are a reasonable evolution, since they have shown to work well with discrete data [11]. They use neural networks to model the probability $p(w)$ of a word w, similar to what n-grams do, but taking profit of the analytical power of neural nets [12].

These models are mostly based on recurrent neural networks (Section 2.4.2), since they are able to consider sequential data dependencies [43]. More specifically, well performing recurrent language models consider the usage of LSTM nets [15] (Section 2.4.3)

and include an attention mechanism in order to detect more complex dependencies between words [42].

Another widely extended architecture to build language models is the Transformer architecture [66]. As explained in Section 2.4.4, the Transformer is the state-of-the-art architecture for sequence analysis tasks like ASR. It features a faster and more refined attention mechanism called self-attention consisting on relating words in the same sequence that allows the Transformer to consider sequences with undefined length with very good results in both long and short term dependencies.

Aside from that, Transformer LMs have shown to perform extraordinarily good in streaming setups [9], achieving slightly better results than LSTM based models with considerably lower latencies.

2.5.2. Acoustic model

In a hybrid ASR system, the acoustic model (AM) is the part in charge of modeling the probability $P(\mathbf{x}|w)$. Historically, the most common approach to this model is a set of Hidden Markov Models (HMM), since it is a structure capable of correctly managing temporal flow.

In order to apply HMMs to ASR, every word w has to be broken down into a sequence of phonemes. This, in turn, has to be decomposed into a sequence of phonemes with context (triphonemes), the latter essentially representing a phoneme alongside their preceding and succeeding phonemes in the sequence. In order to do so, a pronunciation dictionary is needed. This dictionary is simply composed of such representation for each word in the vocabulary.

HMMs can be thought of as a way of model probability distributions over sequences of observations. In the context of ASR, the observations are the acoustic feature vectors. It is assumed that the observations are taken in equal time intervals, so that the observation $\mathbf{x}_t$ has been taken at the time instant t. HMMs are based on the Markov assumptions[49]:

- Each observation $\mathbf{x}_t$ is generated by a state s_t, being this the only active state in the time instant t and that is occult to the observer.
- The active state in the instant t depends only on the active state in the instant $t-1$ (s_t depends only on s_{t-1}).

This kind of models are able to distribute a unary probability mass between all the possible sequences of observations. The probability of a sequence of observations can be expressed as:

$$p_w(\mathbf{x}) = \sum_{s \in S} \prod_{t=1}^{T+1} p(s_t|s_{t-1}) \prod_{t=1}^{T} p(\mathbf{x}_t|s_t) \tag{2.15}$$

Where $p_w(\mathbf{x})$ is the probability of the sequence x yielded by the HMM associated to the word w (in other words, $p(\mathbf{x}|w)$), S is the set of every sequence of states s such as $|s| = |\mathbf{x}|$, s_t is the t-th state visited in the sequence s and $\mathbf{x}_t$ is the t-th vector from the sequence a. The probability $P(\mathbf{x}_t|s_t)$ of the state s_t emmiting the observation $\mathbf{x}_t$ is called "emission probability", and has historically been estimated using Gaussian mixtures [61]. However, recent approaches propose the usage of DNN to model this probability, having

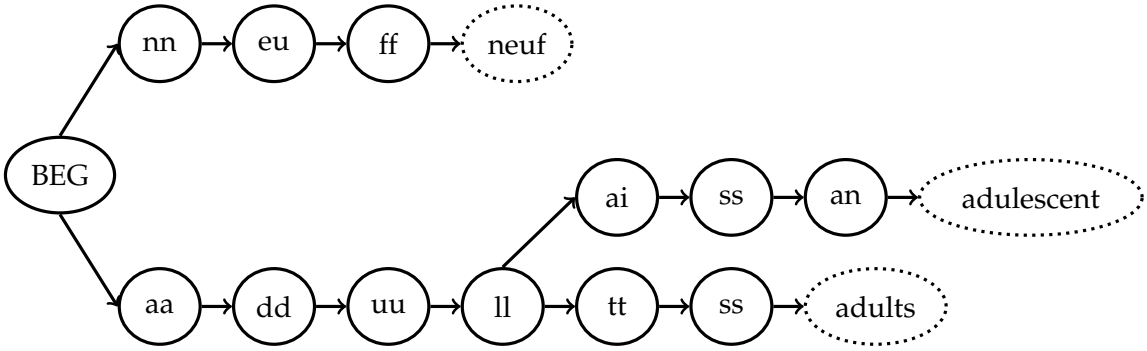


Figure 2.7: Simplified decoder for single word recognition. A simplified language {"adults", "adulescent", "neuf"} is considered.

obtained considerably better results [30]. Still, in order to be able to use DNN in this task, the Bayes' theorem needs to be applied, rewritting the emission probability as follows.

$$p(\mathbf{x}_t|s_t) = \frac{p(\mathbf{x}_t)p(s_t|\mathbf{x}_t)}{p(s_t)} \tag{2.16}$$

Where the DNN is in charge of modeling $p(s_t|\mathbf{x}_t)$, $p(s_t)$ is the prior probability of the state s_t and $p(\mathbf{x_t})$ only depends on $\mathbf{x}_t$ itself, so it can be safely removed from the equation.

Other recent approaches propose neural models in order to estimate $p(\mathbf{x}|w)$ since they don't make assumptions about the data source. More recently, RNNs, more specifically BLSTMs, and the Transformer architecture have recently shown excellent performance in acoustic modeling, being the state-of-the-art for this task [26, 67].

2.5.3. Hybrid decoding

In a hybrid ASR system, the decoding step is where both the acoustic and language models are combined in order to find the most probable sequence of words given the acoustic data. In order to do this, a graph containing all the HMM states is needed. This graph consists of the unfolding over time of the whole set of HMM states, updating the transition probabilities between them on each level according to the knowledge of the language model. The transitions in the decoder graph can either be the transitions between the inner states of the HMMs or the connections between the actual HMMs, called across-word transitions. However, if we simplify the problem to recognition of single words, this graph can be seen as a prefix tree where each state represents a phoneme. Figure 2.7 shows a simplified example of single word recognition.

With this idea in mind, the hybrid decoder can be thought of as a concatenation of prefix trees via across-word connections (i.e. when a word w is recognized, the whole prefix tree is appended to the node of w). Following this idea, Figure 2.8 zooms out the schema in Figure 2.7 to represent multiple word recognition.

As it can be imagined, this complex mechanism can compromise the latency of the system, since querying the language model more than necessary can be too expensive specially in streaming applications. In order to solve this, it is possible to evade unnecessary queries when performing look-ahead operations [45] (consult the LM when a word-node was not reached yet) by making use of static look-ahead tables [33].

The static look-ahead tables are usually build from a simplified version of the used LM. In this case, since an n-gram LM is being used, the static look-ahead tables can be build from an m-gram model with $m \leq n$, or simply a version of the n-gram with less

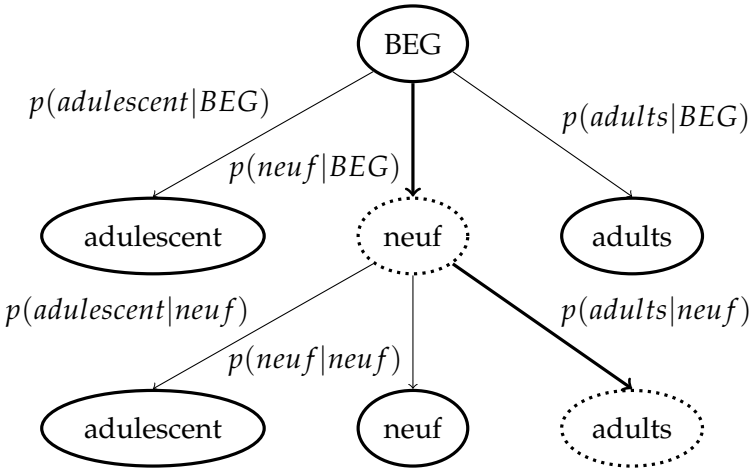


Figure 2.8: Hybrid decoder for multiple word recognition. The language considered is the same as in Figure 2.7 and the sequence recognized was "neuf adults". Probabilities consider a bigram language model and denote the language model factor for the words in each node

entries. This way, the decoder reduces drastically the queries to the "big" LM, the latter being only queried when a word-node is reached.

Aside from this, other upgrades like variance regularization [56], lazy evaluation and language model history recombination [33] allow the decoder to obtain faster and better results.

2.6 Evaluation measures

To evaluate the performance of an ASR system, there are several measures available. Trained ASR systems are evaluated using word error rate (WER) on (manually) labeled development and test speech data.

The word error rate is defined as the minimum edit distance between the correct transcription and the recognized one. It is similar to the Levenshtein distance of edition, allowing insertion, deletion and substitution of words. It is computed as follows:

$$WER = \frac{I + D + S}{|R|} \tag{2.17}$$

Where I, D and S represent the number of insertions, deletions and substitutions respectively, and $|R|$ is the length of the reference transcription.

However, in the specific case of LMs, they are optimized to minimize perplexity (PPL) over a given development set. The perplexity of a given sequence of words $w = \{w_1, ...w_n\}$ can be formally expressed as:

$$PPL(W) = 2^{-\frac{1}{n}\log P(w)} \tag{2.18}$$

Where $-\frac{1}{n}\log P(w)$ is an estimation of the cross-entropy for w when it is sufficiently long. The PPL of the language model can be seen as the estimated number of words that can follow a given word in a sequence. Larger values of PPL mean higher branching, which is non-optimal since the model has to evaluate more words when analyzing a given sequence.

CHAPTER 3
Language model training

This chapter describes how to train count-based and neural language models for streaming ASR. It is structured as follows. First, Section 3.1 introduces the concepts treated in this chapter. Second, Section 3.2 describes in detail the steps needed to train an n-gram LM. Then, Section 3.3 discusses the training process of the Transformer LM (TLM). Lastly, Section 3.4 explains the process of interpolating both LMs.

3.1 Introduction

Two different language models have been trained for this work: an n-gram LM and a Transformer LM. It is important to remember that an n-gram model only calculates n-gram probabilities based on n-gram frequencies observed in a training set, whereas the transformer model implements an attention mechanism that allows it to consider significantly longer histories and, therefore, yield presumably better results.

The process of training the language models is done after the text data preprocessing and consists on two main steps: the training of the n-gram and the transformer LMs and the interpolation of both of them. The final language model consists of a weighted combination of the two trained models that optimizes the development set. Figure 3.1 shows a simplified schema of the sequence of actions to perform in order to develop the final language model.

3.2 N-gram model training

In order to obtain an n-gram model out of a set of training corpora, it is either needed to combine them and work with all data as a unique large corpus, or train individual n-gram models for each text corpus in the training set and linearly interpolate them to minimize perplexity on a given development set.

The procedure of training a n-gram LM out of a set of training corpora, is described as follows. First, a large n-gram LM (hereafter referred to as background model) is trained using a concatenation of all the corpora in the training set. This background model is used as a "generalist" model able to answer most of the queries. Second, an individual n-gram LMs for each corpus is trained. These smaller models are used as "specialized" models that will provide more accurate results in the specific field of their corpus. Finally, these models are linearly interpolated aiming to minimize the perplexity of the development set. This approach results in a more robust model, since the background model will

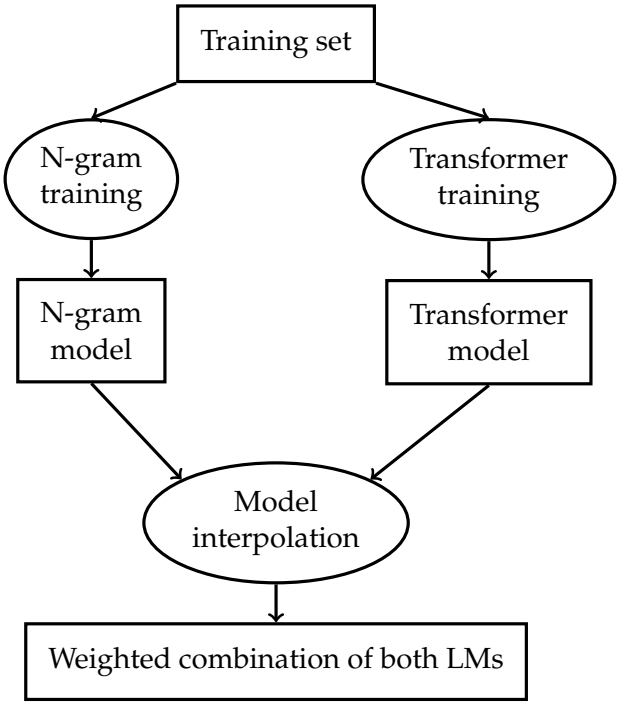


Figure 3.1: Simplified sequence of actions required to train the final LM.

generally obtain a higher interpolation weight than the smaller ones, avoiding uncertain query answers.

Given the schematic description of the target four-gram LM, the sequence of steps taken to build it is the following: First, individual unigram LMs for each training corpus are estimated and linearly interpolated, using as interpolation weights those that minimise PPL on a given development set. The resulting interpolated unigram LM is used to carry out an analysis of single word frequencies in the training data, weighted by their relevance w.r.t. the development set. This analysis is used to define the vocabulary of the final ASR system. Typically, the K most frequent words are selected. This vocabulary reduction is done in order to obtain a balance between the perplexity of the model and its size, which is specially useful for streaming applications. When a smaller vocabulary is considered, all the non-included words are mapped to `<UNK>` and treated as unknown words.

Second, having defined the vocabulary size of the LMs, and hence, of the ASR system, the following step is to train both the background (with the full concatenated corpora) and the smaller corpus-specialized models and interpolate all of them. Again, considering that the final ASR system has to be streaming-ready, this is, capable of dealing with streaming audio signals with tight latency constraints, it is needed to prune the models at this point in order to reduce their sizes and thus improving their query latencies. The pruning is done by removing from the model those n-gram entries with a number of occurrences less or equal to the pruning parameter. After this step, and considering the different pruning parameters, a selection of the pruned models based on their size is interpolated. This selection is chosen considering the sizes of all the individual models in order to obtain a final model with an appropriated size for being used in an streaming environment, this is, finding a model that achieves a good compromise between its size and PPL score.

The interpolation process consists on obtaining a linear combination of all the models that minimizes the perplexity with respect to the development set. It can be thought of as an n-gram model mixture where the only non-fixed parameter is the weight of each

model. Following this approach, the probability of a word w given a history h can be expressed as:

$$P(w|h) = \sum_{m=1}^{M} \lambda_m \cdot p_m(w|h) \tag{3.1}$$

Where M is the number of models, λ_m is the interpolation weight assigned to the m-th model and h is the sequence of words previous to w where $|h|$ is less or equal than three. With this representation in mind, the target of the interpolation is to tune λ_m aiming to minimize the PPL on the development set. This process can be implemented via an Expectation-Maximization [20] (EM) algorithm.

Finally, the last step for building the n-gram LM is to prune the interpolated model. This procedure is similar to the one followed when pruning the corpus-based models in the previous step. In this case, the pruning parameter refers to the effect that an n-gram has on the PPL of the model. That is to say, the pruning process computes the effect that every single n-gram has on the PPL when they are either considered or not. With this idea in mind, the n-grams with an impact on the PPL lower or equal to the threshold value are removed.

With these steps done, the resulting n-gram LM will be used as an external model during the decoding process. However, as explained in Section 2.5.3, the hybrid decoder can use a smaller and faster n-gram LM as look-ahead tables to avoid the latency of querying the bigger model. Considering this, a super-pruned version of the already trained n-gram LM can be built. To build this super-pruned model, the original interpolated model must be pruned with more aggressive values, aiming to reduce its size to a few MB.

3.3 Transformer model training

The transformer models are trained to yield the probability $P(w|h)$ of a word w given its preceding history h by using a self-attention mechanism. The strong aspect of this kind of models is the fact that the self-attention mechanism is applied to a potentially infinite history in a parallel manner, giving them the ability to learn long-term dependencies with better results than RNNs or LSTMs.

The procedure of training a TLM out of a set of training corpora, is described as follows. First, a reduced selection of the training corpora is needed, since the TLM training process is very time-expensive, specially compared to n-grams. This selection can be based on the interpolation weights of each corpora computed in the n-gram training (Section 3.2). This subset of training data can lead to a reduction in the vocabulary size. For this reason, the words in the original training vocabulary that do not appear in this subsection are mapped to `<UNK>`. Then, this data subset is used to train the TLM until it converges, controlling the PPL evolution for the development set. Once the training process is completed, it is possible to compute the average values for the parameters of the TLM using its state in different checkpoints of the training process. This is done to avoid possible loss of between the checkpoints, building a more robust model.

3.4 Model interpolation

After having trained both the four-gram and the Transformer models, the next step is to compute optimum interpolation weights, to be provided to the decoder, as the interpo-

lation of both models probabilities is done on-the-fly during the decoding process. For this, the procedure is equal to the one explained in Section 3.2: first, the perplexity of the models with respect to the development set is computed. Then, the linear combination of the models that optimizes the development set is computed.

CHAPTER 4
Corpora

This chapter describes all speech and text corpora used for training and assessing the acoustic and language models described in Chapter 5. As explained in Section 1.2, one of the goals of this work is to study the contribution of state-of-the-art language modeling technologies to the overall ASR system performance. Considering this, the data described in this chapter is the same used for training the French ASR system developed by the *MLLP-VRAIN* research group under the context of the EMMA European research project [1].

This chapter is structured as follows. First, Section 4.1 exposes the text corpora used as training data for the LMs. Second, Section 4.2 shows the data used to train the AMs developed by *MLLP-VRAIN*'s researchers. Finally, Section 4.3 gives information on the corpora used as development and test partitions for both AMs and LMs.

4.1 Text corpora

4.1.1. Training corpora

To train the LMs, we used the following 15 different data sources:

- **Audiobooks-1**: The Audiobooks-1 dataset is a corpus created from french audiobooks extracted from the Internet. It consists on 18.0K sentences with 258.6K running words.
- **QCRI AMARA** [7]: The QCRI AMARA corpus is an open multilingual collection of subtitles for educational videos and lectures collaboratively transcribed and translated over the AMARA web-based platform. The current release of the corpus contains 20 languages. The french part features 1.4M running words in 139.8K sentences.
- **COSMAT**: The COSMAT corpus is a set of parallel sentences in French and English extracted from scientific thesis abstracts in the HAL open archive. This corpus contains 32.6M running words in 1.7M sentences.
- **DGT-Acquis** [58]: The DGT-Acquis is a family of several multilingual parallel corpora extracted from the Official Journal of the European Union (OJ) in Formex 4 (XML) format, consisting of documents from the middle of 2004 to the end of 2011 in up to 23 languages. It consists on 93.2M running words and 4.9M sentences.
- **Europarl** [36]: The Europarl parallel corpus is extracted from the proceedings of the European Parliament. It includes versions in 21 European languages and was

originally created for training machine translation systems. The corpus features 2.2M sentences and 61.3M running words.

- **EU-TT2**: The EU-TT2 corpora was extracted in the TT2 European project [6] from the Bulletin of the European Union, which exists in all official languages of the European Union and is publicly available in the Internet. 1.1M sentences containing 21.2M running words form this corpus.
- **EUTV**: EUTV is a corpus built from transcriptions of videos from the EUTV website (currently named Multimedia Centre from the European Parliament) [3] and available in 22 different languages from the European Union. The corpus contains 137.0K sentences and 1.2M running words.
- **Giga**: The Giga corpus is composed of parallel sentences in English and French crawled from international websites. It contains 754.2M running words in 22.5M sentences.
- **NC-V8** [8]: News Commentary is a translation corpus from the WMT workshop containing news text and commentaries from the Project Syndicate, provided as training data for the series of WMT translation shared tasks[1] and containing 193.7K sentences and 5.4M running words.
- **OpenSubtitles** [62]: OpenSubtitles is a corpus formed out of documents from the OpenSubtitles initiative website [5]. It contains a total of 327.8M running words and 48.0M sentences.
- **UN**: The UN translation corpus is a English to French and English to Spanish translation dataset provided as training data for the series of WMT translation tasks [13] featuring sessions from the United Nations assembly. It contains 400.9M running words in 12.9M sentences.
- **Voxforge** [65]: Voxforge is an open speech dataset that was set up to collect transcribed speech for use with Free and Open Source Speech Recognition Engines. Currently, it contains data in seventeen languages. The French dataset contains 182.1K running words and 16.4K sentences.
- **TED**: The TED corpus is a set of transcriptions of TED talks from the TED website. It complements the WIT3 corpus by adding transcribed talks up to 2015. It contains 392.3K running words in 47.7K sentences.
- **TEDx**: The TEDx corpus is a set of transcriptions from TEDx talks (unofficial TED events) in several languages up to 2015. It contains a total of 3.9M running words and 477.5K sentences.
- **Wikipedia**: The Wikipedia corpus is a dump from the French Wikipedia [4] from 2015. It contains 372.8M running words i 25.2M sentences
- **WIT3** [14]: WIT3 is the acronym for Web Inventory of Transcribed and Translated Talks. It is extracted from the multilingual transcriptions of TED talks from the TED website[2] from 2007 up to 2013. It is available in 5 languages and the French version contains 2.5M running words and 143.6K sentences.

Table 4.1 summarizes basic raw statistics of all these corpora.

[1] http://www.statmt.org/wmt15/translation-task.html
[2] https://www.ted.com/

Table 4.1: Basic raw statistics of training text corpora.

Training corpus	Sentences	Running words
Giga	22.5M	754.2M
UN	12.9M	400.9M
Wikipeda	25.2M	372.8M
OpenSubtitles	48.0M	327.8M
DGT-Acquis	4.9M	93.2M
Europarl	2.2M	61.3M
COSMAT	1.7M	32.6M
EU-TT2	1.1M	21.2M
News Commentary	193.7K	5.4M
TEDx	430.9K	3.5M
WIT3	143.6K	2.5M
AMARA	139.8K	1.4M
EUTV	137.0K	1.2M
TED	47.7K	392.3K
Audiobooks-1	18.0K	258.6K
Voxforge	16.4K	182.1K
Total	119.6M	2.1G

4.2 Audio corpora

For the training process of the acoustic model used in Chapter 5 as part of the final system, the data used considers a manually transcribed corpus and a set of transcriptions from different sources, grouped by their characteristics. These last grouped sets of data will be considered as individual corpora in order to simplify the section. Table 4.2 contains basic raw statistics for these corpora. The description of the mentioned corpora is the following:

- **Educational data**: this is a collection of conferences and meetings from educational events belonging to different fields. It contains 210 video files with duration between 5 and 15 minutes. It contains a total amount of nearly 39 hours of good quality recordings with different speakers.
- **Entertainment data**: this data belongs to different multimedia entertainment material, television content and domestic recordings. The corpus has a total amount of 200 hours of recordings with lots of speakers.
- **Parliament data**: this corpora contains recording from parliament meetings in 220 audio files and over 420 hours of recordings.
- **Voxforge** [65]: as explained in Section 4.1, Voxforge is an open speech dataset that was set up to collect transcribed speech for use it with FOSS recognition engines. The French part contains 1631 audio files from different speakers, adding up a total of nearly 27 recorded hours.

4.3 Development and test corpora

To evaluate the language models, two corpora were used for building both development and test sets:

Table 4.2: Basic statistics of training audio corpora.

Training corpus	Duration
Educational data	$38h\,59m$
Entertainment data	$399h\,3m$
Parliament data	$421h\,4m$
Voxforge	$26h\,35m$
Total	$885h\,41m$

- **PoliMedia**: The PM dataset is a set of transcriptions of French lessons at PoliMedia. Polimedia is the service for the creation and distribution of multimedia educational content at the UPV. The development partition contains 1h 9m of recorded speech in 8 videos containing 10.7K running words in 1.1K sentences. For testing, it features 52m of recordings in 12 videos with 682 sentences with 6.3K running words.
- **Université de Bourgogne**: The UB dataset is a set of transcriptions of MOOC courses from the Université de Bourgogne. It contains two manually transcribed courses: a wine course and an Internet research course. The wine course, used for the development set, features 18 files adding up 1h 20m of recordings, containing 13.9K running words in 1.4K sentences. The Internet course is used as test partition and contains 1h 57m of recordings in 18 videos, with 1.9 sentences and 19.3 running words.

Table 4.3 contains basic raw statistics of both the PM and UB corpora.

Table 4.3: Basic statistics of development and test data.

Corpus	Set	Sentences	Running words	Duration
PM	Dev	1.1K	10.7K	1h 20m
	Test	682K	6.3K	1h 57m
UB	Dev	1.4K	13.9K	1h 9m
	Test	1.9K	19.3K	0h 52m
Total	Dev	2.5K	24.6K	2h 29m
	Test	2.6K	25.6K	2h 49m

CHAPTER 5
Experiments and evaluation

This chapter describes the experimental setup, the experiments carried out to build and optimize our proposed ASR system, and the empirical results obtained in each development and assessment step. At the end of the chapter, the resulting system will be compared with the French ASR hybrid system developed in 2017 by the *MLLP-VRAIN* group. The LMs developed in this work will be combined with an acoustic model developed by *MLLP-VRAIN*'s researchers based on BLSTMs.

This chapter is structured as follows. First, Section 5.1 describes the toolkits used to train the LMs and recognize with them. Second, Section 5.2 shows the preprocessing steps carried out to clean the data. Third, Section 5.3 exposes the process of training the n-gram LM and the TLM, as well as their interpolation. Then, Section 5.4 discusses the integration of the trained LMs into the hybrid decoder, as well as the process of optimizing decoding. Finally, Section 5.5 compares the built ASR systems to the baseline 2017 French ASR system.

5.1 Toolkits

In the process of building our LMs, three main tools have been used: KenLM, TLK and Fairseq. First, KenLM [29] is an open-source language modeling toolkit developed by Kenneth Heafield consisting of a set of C++ executable programs that is widely used in speech recognition. In this work, KenLM is used to build n-gram models out of the data described in Section 3.2, interpolate them and evaluate the resulting model. For this, the following executable binaries from the toolkit are used:

- `lmplz`, which stands for *"language model please"*, is used to train n-gram models. It reads the training data from the standard input and calculates the probability for each n-gram. Several options are available, but the most important ones are `-o`, which specifies the n-gram order, and `-prune`, which removes the n-grams with a number of occurrences less than the integer specified.

- `query` is used to calculate model probabilities and perplexities of a set of sentences, typically those from the development dataset. It takes the model as an argument and the sentences or queries through the standard input. The output consists of the log-probabilities for each word in a sentence ($\log_{10} p(w|h)$) alongside the order of the n-gram matched and the id of the word in the LM, as well as the perplexity of the model at the end of the output.

The rest of the steps regarding the n-gram model (calculate the interpolation weights, prune the resulting models...) were made using custom scripts that complement KenLM developed by the *MLLP-VRAIN* group.

Second, The transLectures-UPV Toolkit [19] (TLK) is a state-of-the-art toolkit developed by the *MLLP-VRAIN* research group at the Universitat Politècnica de València (UPV) under the context of the transLectures project [57]. It is a toolkit aimed at developing software capable of transcribing speech that allows to preprocess acoustic data, train acoustic models and decoding. This is the toolkit used for building the winner ASR systems for both the 2018 and 2020 editions of the Albayzin-RTVE Speech-To-Text Challenge [34, 32]. TLK was used to transform the KenLM models (in `arpa` format) to the TLK intern binary format, to build the lookahead tables used in decoding and to perform the decoding step. For this purpose, the following tools have been used:

- `tLlmformat` is a tool used to format a language model to the internal TLK format used for language models. It is used via command-line, receiving the format of the input file as an argument, and the path for the input (`-i`) and output (`-o`) files.
- `tLmksltfs` is used for building the look-ahead tables. It is also used via command-line, receiving a language model in TLK format, a lexicon file and a tied-phonemes list(these two files are explained in detail in Section 2.5.2), and generates a `.sltfs` file containing the look-ahead table.
- `tLtask-recognise` is the tool used for performing the decoding task. It receives all the models involved in the recognition, apart of the decoding parameters.

Finally, Fairseq [47] is a toolkit developed by Facebook AI Research that allows custom model training for language modeling, among other text generation tasks. It is based in PyTorch and supports multi-GPU and multi-machine training. Fairseq was used for training the Transformer LM (TLM). Two main tools have been used for this purpose:

- The `fairseq-preprocess` tool prepares the train and development (validation) data, receiving them as two separate input files (one for each task) and generating all the necessary files to start training.
- The `fairseq-train` tool is used to train the TLM. It receives as input the directory containing all the files generated by `fairseq-preprocess`, the options of the model and the process of training (architecture, topology, criterion, learning rate...).

5.2 Data preprocessing

The data used for training the language models is described in chapter 4.1. However, before starting to train the models, some preprocessing steps must be performed in order to normalize the data. These preprocessing steps consist on removing special characters, converting numbers to text and remove punctuation marks. Essentially, no characters other than non-capitalized letters from the French alphabet must appear on the data files. This implies that, since the French alphabet contains letters which do not appear in the strict Latin alphabet, the text files must be written in Unicode format. In order to clean the text files, several main steps were performed:

First of all, each dataset needs a specific preprocess consisting in removing all sort of non-alphabetical characters (for example, there were musical notes, copyright symbols and smart quotes, among other symbols) or special sequences such as xml code lines.

Table 5.1: Train corpus statistics

Training corpus	Sentences	Running words
Giga	22.5M	700.9M
Wikipeda	25.1M	393.6M
UN	12.9M	372.5M
OpenSubtitles	48.0M	263.9M
DGT-Acquis	4.2M	102.0M
Europarl	2.2M	55.7M
COSMAT	1.6M	32.6M
EU-TT2	1.0M	20.3M
News Commentary	193.7K	4.9M
TEDx	430.0K	3.5M
WIT3	143.5K	2.5M
AMARA	138.5K	1.2M
EUTV	137.0K	1.2M
TED	47.5K	387.1K
Audiobooks-1	18.0K	258.6K
Voxforge	16.4K	181.7K
Total	118.6M	2.0G

This was achieved using the UNIX `sed` utility, replacing special characters either by their representation in Unicode, or blank characters if there is no representation.

The following step is to normalize the text. For this, two main substeps were performed: First, it is needed to expand abbreviations (this is *dr.* $\rightarrow$ *docteur*, *m.* $\rightarrow$ *monsieur*...). For this, a script containing the most used abbreviations in French was created to substitute each one of them with their corresponding expansion. Then, it is needed to convert numbers to words (transliterate numbers). This was done using the Python library `num2words`, which offers support for the French language, and extending its functionality to cover other cases such as decade numbers, ordinals, etc.

In order to correctly perform these steps, it is necessary to previously tokenize the text. Tokenize is the process of extracting each individual word as a token in order to be able to treat each one of them separately. It is also usually helpful to process some characters like apostrophes before it, since they may interfere with the tokenisation. In this case, apostrophes are treated by replacing them with special characters before the tokenisation and then being replaced back at the end of the preprocess. It is also convenient to remove all punctuation marks before performing the normalization steps, since they could interfere with abbreviations (most of them end in a point) or with number notation, more specifically in century notation.

Finally, a cleaning step is performed. In this last step, it is important to remove all words with non alphabetical symbols, replace all special characters removed in the tokenisation step, trim the text and remove empty lines to obtain a clean dataset ready to be used for LM training. Table 5.1 shows the training statistics after having followed these preprocessing steps.

Next, smaller, domain-related corpora are combined (joined) in order to reduce both the training time and the number of trained models. In this case, we generated two corpora combinations: On the one hand, Audiobooks-1 and Voxforge were combined since they are the smallest corpus in the training set, and both are related to text derived from audiobooks. On the other hand, TED and TEDx were also combined since they

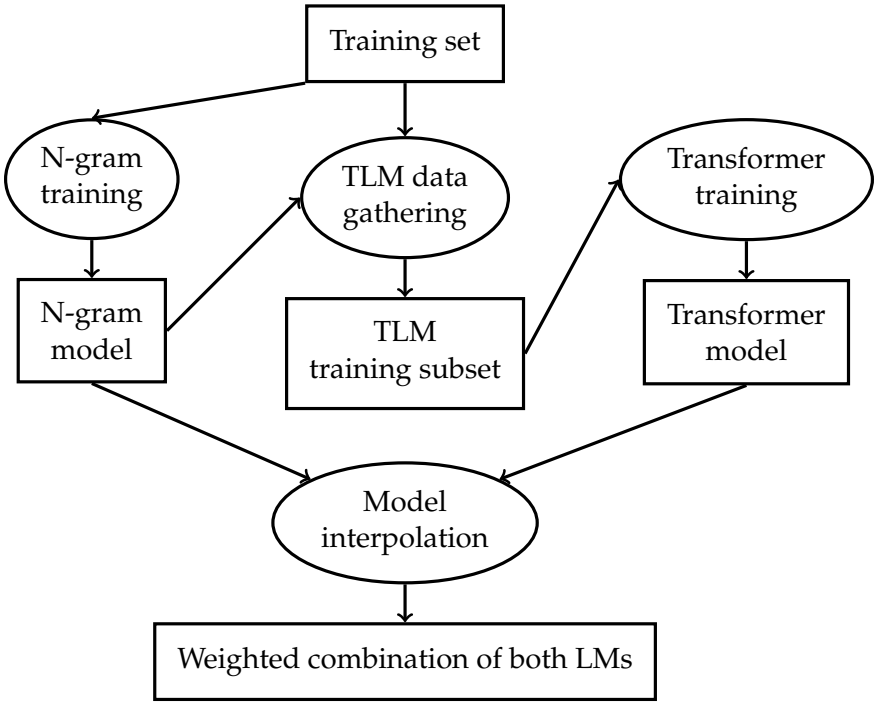


Figure 5.1: Simplified sequence of actions needed to train the final LM.

both correspond to either official or unofficial TED talks. Table 5.2 shows statistics for both combinations.

Table 5.2: Combined corpus statistics

Training corpus	Running words	Sentences
Audiobooks+Voxforge	440K	34K
TED+TEDx	3.9M	478K

5.3 Language model training

In this work we trained two different LMs: a 4-gram LM and a TLM. Both LMs can theoretically be trained in parallel. However, due to the computational cost of training the TLM, a reduced subset of training data must be considered to alleviate this concern. This subset is selected following a specific criteria that considers the information extracted from the n-gram LM training process, as it is explained later on in Section 5.3.2, so it prevents the possibility of running both tasks in parallel. Figure 5.1 provides an updated version of Figure 3.1 on the sequence of actions performed to obtain the final LM.

5.3.1. 4-gram LM

As explained in Section 3.2, the first step to train the 4-gram model is to decide the vocabulary of the final system. To do this, first, we trained a unigram model to sort all words from the training data by their importance w.r.t. the development data, instead of, e.g. sorting them by their absolute frequency. Then, different vocabulary sets are defined by selecting top-ranked words fitting different vocabulary sizes. Finally, out-of-vocabulary (OOV) ratios of development data are computed for each vocabulary set. OOV ratio is the percentage of words present in the development data not included in the vocabulary set, this is, those words that the final system would not be able to recognize.

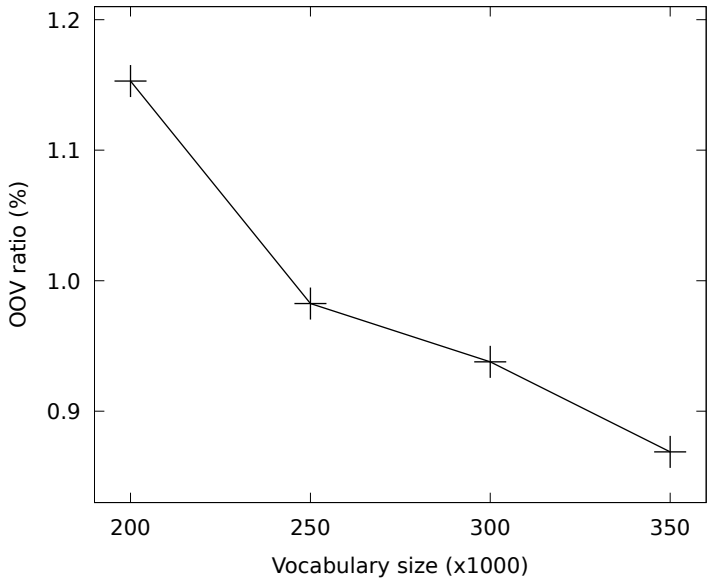


Figure 5.2: OOV ratio on the development set for the different vocabulary sizes (in thousands of words) considered.

In this case, the full vocabulary size of the whole training data comprises 4 667 840 unique words. Considering that the resulting system aims to deliver good performance under an streaming environment, it was mandatory to significantly shrink the vocabulary size. We explored vocabulary sizes $|V| = \{200, 250, 300, 350\}$ (in thousands of words, K). Figure 5.2 shows the evolution of the OOV ratio (vertical axis) w.r.t. the size of the different vocabulary sets considered (horizontal axis).

The best candidates were $|V| = \{250, 300\}$, but due to the small difference between their OOV ratio values, it was considered not worth increasing the vocabulary by 50K words for just an improvement of 0.05%. Hence, the final decision fixed the vocabulary size in $K = 250$, obtaining an OOV ratio value of 0.98%.

The following step was to train all the 4-gram models (corpus-specific models and background), with some slight model pruning by removing the n-grams with a number of occurrences lower than the pruning parameter in order to reduce their size. This step was done using the KenLM tool `lmplz`, which allows to prune the models whilst training them. The pruning values considered in this step for every model was $p = \{1, 2, 5\}$.

After this step, the interpolated model is built from a selection of the trained models considering their weight. This decision is normally taken based on experimental results, but due to the time constraint and considering that the sizes of the models were not big enough to represent a threat on the final system's latency, the interpolation was done considering all the models pruned to one occurrence (p=1). The interpolation was done using a custom tool `interpolate` developed by the *MLLP-VRAIN* group that implements the interpolation process explained in Section 3.2 via an Expectation-Maximization (EM) [20] optimization algorithm, rather than using the KenLM tool (which uses a $\log p(x)$ interpolation approach). Table 5.3 shows the optimum interpolation weights for each individual 4-gram model.

The resulting model from the previous step had a size of 3.1GB, which exceeds the standards for an streaming n-gram model. Based on previous knowledge from the *MLLP-VRAIN* group, the target was to reduce the size nearly to half. For this reason, we performed a second pruning process in order to reduce the model size. This pruning process is explained in Section 3.2 but, in essence, it removes n-gram entries considering their ef-

Table 5.3: Optimal interpolation weights computed for each quad-gram model with respect to the development set.

Language model	Interpolation weights (%)
Wikipedia	33.8
Background	19.5
Amara	18.7
TED+TEDx	16.7
Cosmat	7.4
Giga	2.5
Opensubtitles	1.1
DGT	$9.0 \cdot 10^{-1}$
Audiobooks+Voxforge	$4.0 \cdot 10^{-2}$
Europarl	$8.0 \cdot 10^{-4}$
Wit3	$1.4 \cdot 10^{-5}$
EUTV	$1.0 \cdot 10^{-6}$
UN	$1.1 \cdot 10^{-12}$
EU-TT2	$7.8 \cdot 10^{-17}$
Ncv8	$3.2 \cdot 10^{-17}$

Table 5.4: Threshold pruning values that provided the best results against the size reduction achieved with them.

Pruning threshold value	Size reduction
$2 \cdot 10^{-11}$	41.94%
$5 \cdot 10^{-11}$	54.84%
$8 \cdot 10^{-11}$	61.29%
$2 \cdot 10^{-10}$	72.16%

fect on its PPL. Table 5.4 shows the best performing studied threshold values for the second pruning process and their effect on the size of the model.

After studying the effect of the threshold values on the size of the models and considering the values that yielded a model with the desired size, the decision was made by studying their effect on the model's PPL. Figure 5.3 shows the evolution of the PPL compared to the pruning value (and thus the size) of the models considered in table 5.4. At the light of the results, we selected the threshold value $p = 5 \cdot 10^{-11}$, as it obtains the minimum value for the PPL while yielding a half-sized model.

The last step was to create a significantly smaller model that will be used to generate the static look-ahead table needed by the system decoder. To build this super-pruned model, we considered values of the order of $1 \cdot 10^{-8}$. Table 5.5 shows the resulting model size for the best performing values studied alongside of their PPL. After studying the resulting models, the final super-pruned model was with a pruning threshold of $4 \cdot 10^{-8}$.

Completing the 4-gram model training took approximately three weeks on quad-core Intel i7 CPUs with 64GB of RAM. The original interpolated model, and its pruned and

Table 5.5: Threshold pruning values that provided the best results alongside with the resulting model size and the resulting PPL.

Pruning threshold value	Model size	PPL
$4 \cdot 10^{-8}$	36M	320
$5 \cdot 10^{-8}$	31M	327

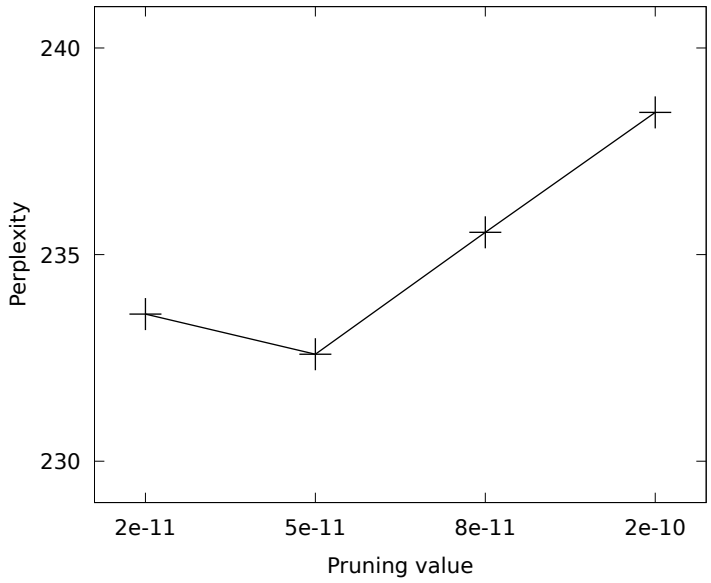


Figure 5.3: Evolution of the models' PPL values against their pruning values. Lower is better.

super-pruned versions had final sizes of 3.1GB, 1.4GB and 36MB while showing PPL values of 233, 235 and 320 in the development set, respectively.

5.3.2. TLM training

As explained in Section 3.3, before starting the training process it is necessary to preprocess data. However, when dealing with Transformer LMs, the *MLLP-VRAIN* group has historically encountered difficulties for training with amounts of data exceeding more than 1G running words. Following their advice and considering time constraint, we selected a subset of 1G from the whole training data, that comprises 1G running words. This selection process consists on selecting the highest ranked corpora in the interpolation of the 4-gram models (see table 5.3) and concatenating them in order to obtain a set of the desired size. The selection considered the Wikipedia, Amara and TED+TEDx corpora in their integrity, which comprised a total of 398.7M running words. The remaining data to reach 1G running words was randomly selected from the remaining unselected corpora. However, this reduced training set can lead to a reduction on the system vocabulary, causing a bigger chance of finding unknown words. In order to avoid this problem, after the training process, Fairseq equally distributes the probability assigned to `<UNK>` among the words that do not exist in the TLM training vocabulary. This can cause an increase on the PPL shown in the training process.

The next step was to preprocess this subset of training data. In this process, we built a dictionary containing the TLM vocabulary and converts the data into a binary format, making it easier for the training process to read it. This process is done using the Fairseq tool `fairseq-preprocess`.

The following step is to start training the TLM. This was done using the Fairseq tool `fairseq-train` with a custom variance regularisation implementation [56], that receives as parameters the full set of training parameters (data location, model topology, learning rate...). The trained TLM had a configuration of 24 decoder layers, 512 cells per layer, 4096-unit FF-DNN, 12 attention heads and an embedding of 768 dimensions. This configuration was based on prior knowledge and experience from MLLP-VRAIN researchers. Due to time constraints, no other configurations were explored. The process of train-

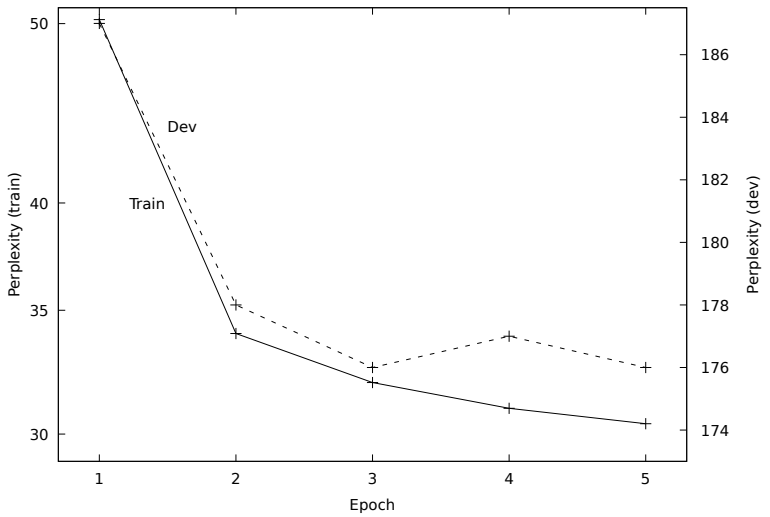


Figure 5.4: PPL evolution through the epochs of the TLM training process on both train and development sets.

ing the TLM took approximately two weeks running on two NVIDIA RTX 2080ti GPUs. Figure 5.4 shows the PPL evolution during the training process for both train and development sets.

Once the training process is completed, Fairseq leaves a certain number of checkpoints corresponding to the state of the model in a specific moment of the training process specified as an argument in the `fairseq-train` order. In this case, 10 checkpoints were saved, which were used to compute an average TLM by computing the average parameters of each checkpoint. This averaging process was performed using a python script provided by Fairseq named `average_checkpoints.py`.

The next step is to normalize the `<UNK>` probabilities to work with the full system vocab. This was done via a custom script made from the *MLLP-VRAIN* group named `eval_lm.py` that receives the full vocabulary size as an argument. Aside from that, and since the TLM aims to provide good results under an streaming context, this script also allows to limit the history length considered by the TLM by receiving it as an argument. We explored history lengths $|h| = 5, 10, 20, 40, 60, 80$. Figure 5.5 shows the PPL results for the studied values.

In this case, the PPL values were computed after normalizing the `<UNK>` probability value. As the results showed, history lengths over $|h| = 10$ provide the same value. The explanation of this phenomenon is that the average length of the sentences in both training and development sets are near 10. Considering this, history length values over $|h| = 10$, the TLM can't extract better results, since the history length considered is longer than the actual sentence. The final decision fixed the TLM history length in $|h| = 10$. The final TLM scored an average PPL of 187 for the development set with the full system vocabulary size.

5.3.3. LM interpolation

After having both the 4-gram model and the TLM, an interpolation of both was computed. This interpolation follows the idea described in Section 5.3.1 and was performed using a custom python script named `compute_mix_weights.py` that implements the EM algorithm explained before. This interpolation can be used in the recognition step pro-

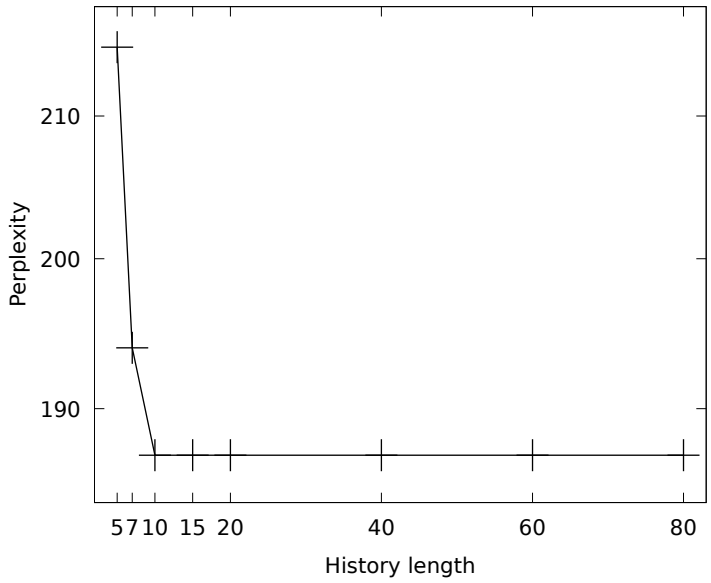


Figure 5.5: PPL evolution over TLM history length

Table 5.6: Interpolation weights that optimize the development test, rounded to the first decimal.

Development task	Interp. weights % (4-gram / TLM)
PM	41.0 / 59.0
UB	40.5 / 59.5
PM + UB	40.7 / 59.3

viding the TLK toolkit with both models and the interpolation weights. Table 5.6 shows the optimum interpolation weights that minimize the PPL on the development set.

Table 5.7 shows and compares PPL scores of the 4-gram LM, the TLM, and the interpolation of both models.

Looking at the results, the best LM, in terms of PPL, is the interpolation of the 4-gram and the Transformer LMs. The explanation of this phenomenon is that the TLM is a way more powerful model than the n-gram, but it was trained using only 1Gwords, against the nearly 120Gwords used for the n-gram model. This means that, in the overall cases, the TLM has the ability to reduce the possible paths better than the n-gram, but there are some cases where words that do not appear in the TLM training vocabulary are seen in the development test. In these less probable cases, the n-gram model still has the knowledge to manage the sentence. Considering this, the interpolation is able to manage the overall cases as good as the TLM, but thanks to the n-gram model, it does not have the problem where words that were not seen in the past appear in the development set as much as the TLM alone does.

5.4 Integrating LMs into the hybrid decoder

In this section we integrate the proposed LMs into the hybrid decoder to build complete ASR systems. More specifically, we combine our LMs with a brand-new BLSTM-HMM acoustic model trained by *MLLP-VRAIN* researchers.

The complete ASR system is assembled when running the TLK's decoder via the `tLtask-recognise` command-line tool. This tool receives both the AM and LM and performs the decoding step. The decoder features many decoding parameters that can be

Table 5.7: PPL comparison for both final models and their interpolation on the development set.

	PPL		
Language Model	PM	UB	Total
4-gram	232	236	235
TLM	187	186	187
4-gram + TLM	155	150	152

tuned to slightly improve the overall system's performance. From those, we highlight the following:

- **GSF**: The *Grammar Scale Factor* (GSF) is used to scale the weight of the LM with respect to the AM in the decoding step. It matches the α factor in Equation 2.7
- **LMHR**: The *Language Model History Recombination* (LMHR) is used to recombine hypotheses that share the same last words of their histories. More precisely, for a given LMHR $= N$ two different LM histories w_1^M and $w_1^{'L}$ are recombined if $w_{M-N+1}^M = w_{L-N+1}^{'L}$. This forces the decoder to consolidate word prefixes and allows a faster decoding, since the history of the TLM is not limited in length and it has to be processed in every iteration.
- **PSF**: The *Prior Scale Factor* (PSF) is used to scale the prior probabilities $p(s_t)$ of the HMM states learned by the AM.
- **LA**: The *Look-Ahead* (LA) window parameter refers to the "future" window size that the AM needs to yield a probability. This window is useful when dealing with BLSTM or Transfomer AMs, since it allows to consider a brief period of time in the future to look for relevant information for a given sample.
- **BEAM**: The *BEAM* parameter represents the maximum difference between the scores of the most and least probable active hypotheses. If the lowest ranked hypothesis differs in score by a bigger value than the BEAM, it is automatically discarded.
- **HP**: The *Histogram pruning* (HP) window parameter refers to the "future" window size that the AM needs to yield a probability. This window is useful when dealing with BLSTM or Transfomer AMs, since it allows to consider a brief period of time in the future to look for relevant information for a given sample.
- **LMHP**: The *LM Histogram pruning* (LMHP) parameter controls the number of new LM histories that will be expanded in each time frame. This allows to reduce the number of queries to the LM.

In this case, the colleagues that developed the AM provided optimal values for the LA and PSF parameters. Since these two parameters mainly affect the AM, and in order to bound the experimental work volume, the study of those will be omitted, using the provided values (LA $= 49$, PSF $= 0.9$). Parameters BEAM, HP and LMHP will be also omitted due to time constraints and will be set to BEAM $= 160$, HP $= 7500$, LMHP $= 60$ based in previous knowledge from the *MLLP-VRAIN* group. Hence, in this work we focused only on the optimisation of the GSF and LMHR parameters for each ASR system.

We started the study by considering the BLSTM AM alongside the super-pruned n-gram LM used as look-ahead table. Since the LM is a four-gram LM, LMHR can't be different than three, as the histories considered by this kind of model have a fixed length

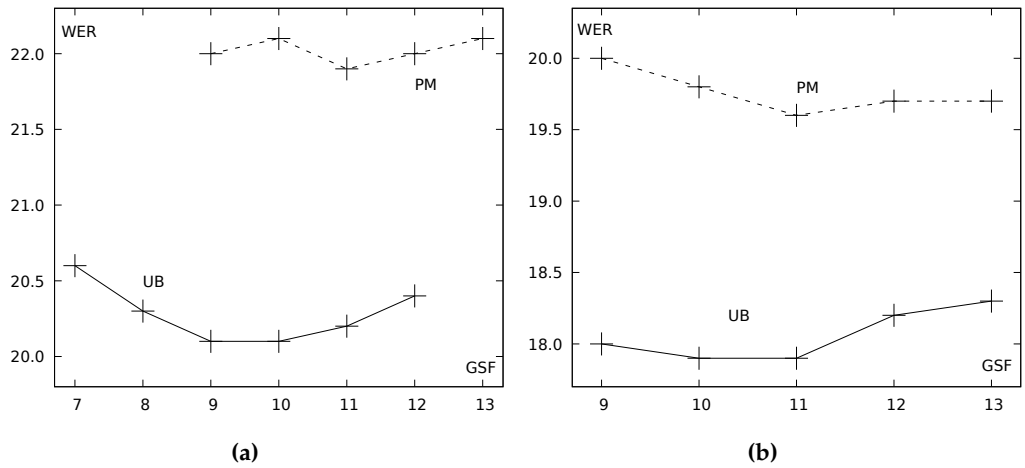


Figure 5.6: GSF exploration for the both ASR systems that include the big and small n-gram LMs. The left figure plots the WER (vertical axis) computed with the system including the small n-gram LM against the GSF parameter (horizontal axis) w.r.t the UB and PM development sets, while the right figure does the same for the system built with the big n-gram LM.

of $|h| = 3$. Figure 5.6a plots the WER scores as a function of the GSF parameter for both PM and UB development partitions. At the light of the results, the PM set had a clear minimal value on GSF $= 11$. Regarding the UB set, both values GSF $= \{9, 10\}$ had the same WER score, so we decided to continue with GSF $= 9$. Next, we moved to the ASR system that incorporates the big 4-gram model. We proceeded exactly as before: we only studied the effect of GSF on the WER since LMHR needs to be equal to three. Figure 5.6b plots the resulting WER scores as a function of the GSF values considered for the UB and PM development partitions. Considering the results, we selected GSF $= \{11, 10\}$ for PM and UB, respectively.

For the TLM, the procedure was the similar: in this case we fixed LMHR $= 9$ and explored a set of GSF values, resulting in a best value of GSF $= 9$ for both development sets. Figure 5.7a shows the WER evolution w.r.t. the GSF parameter in both UB and PM development sets. Once an optimal GSF was found, and considering that the TLM is able to consider histories longer than 3, we fixed GSF $= 9$ and explored the LMHR. Figure 5.7b shows the WER evolution w.r.t. the LMHR parameter in both UB and PM developing sets.

Finally, we consider the optimization of the decoding parameters of the ASR systems that includes a linear interpolation of both the n-gram LM and the TLM. However, in this case, we fixed the LMHR to the best value achieved in the previous step, since this parameter tends to affect similarly in different configurations of the same model. Aside from that, as explained in Section 5.3.2, we realized that the average length of the sentences was near 10 in both the development and the training sets. Considering this, higher values for LMHR would behave similarly in this case (but consuming more computational resources). With this consideration, we studied the GSF parameter effect on the interpolation of both LMs. Figure 5.8 shows the WER evolution w.r.t. the GSF parameter for the interpolation of both LMs on both UB and PM development sets.

5.5 Assessment and comparison of ASR systems

In this section, we perform a final assessment of all ASR systems built in this work by measuring their performance in terms of WER computed over the test datasets (PM and

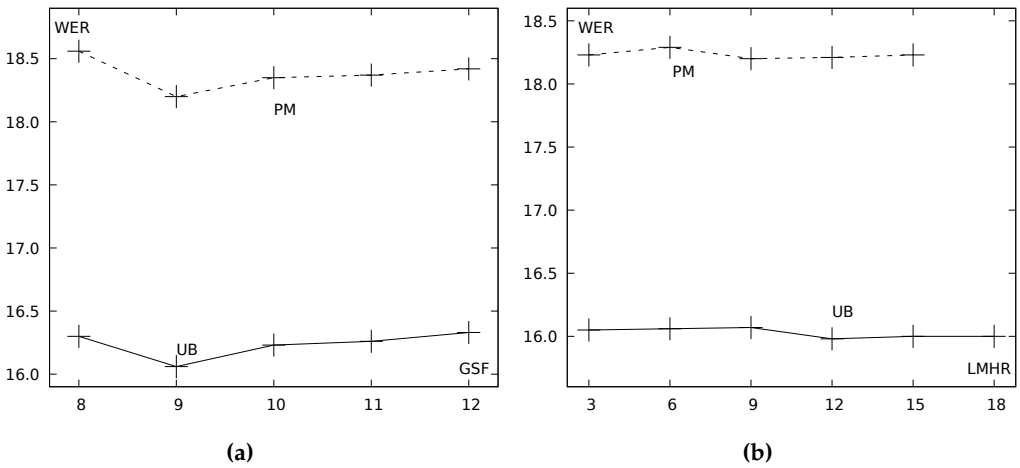


Figure 5.7: GSF and LMHR exploration for the TLM. The left figure shows the effect on WER (vertical axis) of the GSF parameter (horizontal axis) for the UB and PM development sets, whilst the right figure does the same for the LMHR parameter (horizontal axis).

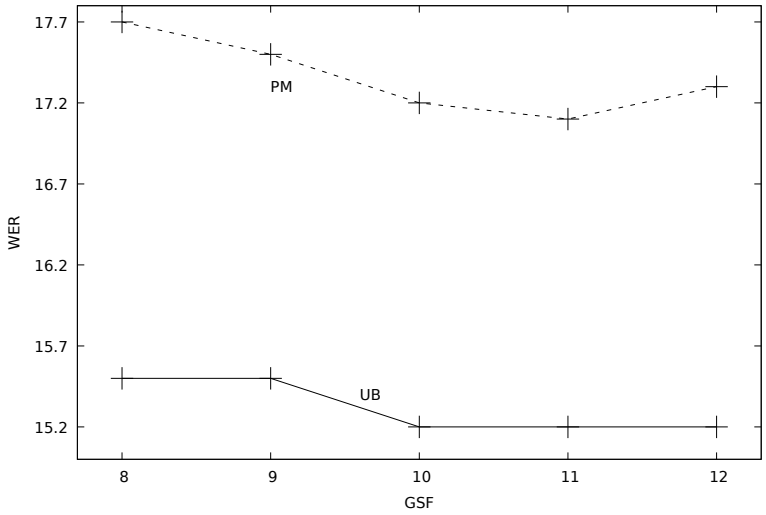


Figure 5.8: GSF exploration for the interpolation of both LMs.

UB). Then, we compare these figures with those yielded by the former, baseline French ASR System developed by the *MLLP-VRAIN* research group in June 2017 to gauge the contribution of the technological upgrades that convert this system a state-of-the-art one.

The baseline system is a French ASR system developed by the *MLLP-VRAIN* group under the context of the EMMA European research project [1], and then upgraded on June 2017 for a Streaming ASR Proof-of-Concept project commissioned by the European Parliament [2]. On the one hand, the AM of the baseline system is composed by two hybrid FF-DNN/HMM acoustic models: a standard model and an fCMLLR model. On the other hand, the baseline system LM incorporates an interpolation of several 4-gram models and a standard RNNLM. The recognition process considers a two-pass approach: first, a recognition pass uses the standard AM to obtain an early transcription of the input speech. Then, the fCMLLR AM and the 4-gram LM are used to perform a second decoding process, generating lattices that contain the most probable hypotheses. Finally, the RNNLM is used to rescore those lattices and obtain the most probable hypotheses, which is considered the final transcription. This scheme is usually referred to as fCMLLR speaker adaptation [23, 59].

Table 5.8: WER % results for the testing process of our ASR systems compared to the baseline system from June 2017. Our ASR systems are represented by adding the LMs developed in this work to the BLSTM AM provided by *MLLP-VRAIN*. Results on the development partitions of UB and PM have been added to complete the comparison.

	PM		UB	
ASR System	Dev	Test	Dev	Test
Baseline (June-2017)	22.1	24.1	19.0	16.5
BLSTM (+LA table)	22.3	22.2	19.8	18.0
+ old n-gram	20.5	20.7	17.4	16.6
+ n-gram	20.1	20.2	17.6	16.3
+ TLM	17.7	18.9	**15.1**	14.9
+ n-gram	**17.1**	**18.6**	15.2	**14.6**

Our ASR systems are composed of the BLSTM AM provided by *MLLP-VRAIN* researchers plus the different LMs developed in this work (small 4-gram LM, big 4-gram LM, TLM and the interpolated TLM plus 4-gram). We are going to perform an evaluation of our four ASR systems on the test set and compare the results with the baseline system. Also, we will add a sixth system composed by the new BLSTM AM and the old LM, provided and optimized by *MLLP-VRAIN* researchers. This old LM is a 4-gram LM trained with the same data as the one in this work but with the SRILM toolkit [60] in 2017, following a different strategy due to the toolkit's particularities.

The recognition process was performed in all the cases using the tLtask-recognise command-line tool from TLK using both PM and UB test sets. Table 5.8 summarizes the WER figures obtained for all these 6 systems, including also those computed for the two development sets and the average result considering the four tasks.

As it can be seen, the baseline system is outperformed by all ASR systems, except by the one that does not incorporate any external LM, apart from the look-ahead table, which behaves similarly or better in PM, but worse in UB. A relevant aspect of this comparison is that, due to the two-pass decoding process, the baseline system can only be used in an offline context, whilst all ASR systems developed in this work are streaming-ready. This feature is achieved by performing the LM rescoring during decoding every time a word is recognised, which also allows the system to obtain better hypothesis, since they are based in more accurate transcriptions than if the rescoring was done in a second pass, as the baseline system does.

Considering the results, we can say that the baseline system (two-pass decoder, two AMs, two LMs) performs similar to the new BLSTM AM with the super-pruned n-gram LM used as look-ahead tables, as they achieve similar WER scores overall. This fact shows that the BLSTM AM is a lot more powerful than the baseline's AM, since it matched the baseline results in an unfair comparison in terms of language modeling (baseline's 4-gram and RNNLM vs our system's look-ahead table alone).

Also, we can see that the new n-gram LM generally outperforms the old LM, only performing worse on UB development set by 0.2 points. An important aspect of this comparison is that each system features a look-ahead table derived from their own n-gram LM. Aside from that, it is important to consider that the new LM was trained with a considerably bigger vocabulary. This fact can rise the PPL of the LM (and thus the WER of the ASR system), but it allows the LM to perform better in out-domain situations.

However, the biggest improvement comes when the TLM is considered. In this case, the WER improvement over the baseline LM stands at over 2 points in almost every set, being UB test the only one below this difference mark. Besides of this, the final TLM

performance was under the expectations, since the expected improvement was higher. This can be due to lack of in-domain data in the training corpus.

Finally, the addition of the interpolation of both the n-gram LM and the TLM represented a shy improvement over the TLM on its own, being its advantage in performance under 1 point in almost every task. However, there is a case (UB-development) where it provides a worse result (0.1 points) than the system with only the TLM.

All-in-all, the best performing system is the one conformed by the BLSTM AM and the interpolation of both 4-gram LM and TLM, which outperforms the baseline by a 29.6% in the best case (PM test) and a 13.0% in the worst case (UB test) and, at the same time, enables its usage under streaming scenarios.

CHAPTER 6
Conclusions and future work

This work has exposed the process of building and integrating state-of-the-art French language models built from scratch into streaming-ready hybrid ASR systems. To do this, we used state-of-the-art software tools such as KenLM, Fairseq and TLK, and a huge collection of monolingual French text data comprising up to 2 billion running words. Finally, the resulting LMs were combined with an external AM provided by *MLLP-VRAIN* researchers in order to evaluate the full system and compare it to the previous French ASR system developed by them five years ago.

With respect to the language models, two types of models were trained: a 4-gram model and a Transformer model. In experimental results, the TransformerLM provided better results than the n-gram model (about a 20% of relative improvement). Furthermore, the interpolation of both models slightly outperformed the TLM on its own.

Following the results from the comparison between the best ASR system developed in this work and the baseline system, it can be said that the goal of improving the existing results was achieved, since the best model considered in this work achieves a WER improvement of 24.4% w.r.t. the baseline model. Aside from that, the new ASR system is prepared to perform well under streaming conditions.

This work leaves many aspects that can be further developed and are open for improvement. Some of them are:

- To tune the decoding parameters specially to optimize the system for the streaming environment. The system developed in this work is steaming-ready, since the decoding step does not require the whole input signal in order to recognize. However, the comparisons made in Section 5.5 were performed in off-line mode.
- To study the performance of the PSF, LA, BEAM, HP and LMHP parameters, in combination with the already studied GSF and LMHR. Due to time constraints, these combinations were omitted and we focused on studying the LM-specific parameters. However, a nested exploration of all these parameters in combination could have further improved performance results.
- To refine the text data preprocessing step. Considering time constraints, our best effort ended with a suboptimal solution. The text preprocessing tools developed in this work consists only on rules that extracted the superficial and general "impurities", rather than studying every corpus in order to obtain the cleanest preprocessed version possible.
- To consider other training corpora exhibiting a higher average sentence word length than the ones used in this work to further exploit the expressive power of Transformer LMs in terms of long-term history attention.

- To consider other development and test sets covering domains more related to the CERN, such as particle or nuclear physics.
- To explore impact on performance of topology parameters of the Transformer, such as the number of encoder layers or the number of attention heads.

Bibliography

[1] EMMA. https://project.europeanmoocs.eu. Accessed on 17/06/2022.

[2] European Streaming ASR proof-of-concept project. https://www.mllp.upv.es/mllp-presentation-on-multilingual-real-time-speech-to-text-at-the-european-parliament/. Accessed on 17/06/2022.

[3] Multimedia Centre from the European Parliament. https://multimedia.europarl.europa.eu/en. Accessed on 17/06/2022.

[4] Multimedia Centre from the European Parliament. https://fr.wikipedia.org/.

[5] Opensubtitles initiative. http://www.opensubtitles.org/. Accessed on 17/06/2022.

[6] TransType2 European project. http://cordis.europa.eu/project/rcn/71419_en.html. Accessed on 17/06/2022.

[7] Ahmed Abdelali, Francisco Guzman, Hassan Sajjad, and Stephan Vogel. The AMARA Corpus: Building Parallel Language Resources for the Educational Domain. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1856–1862, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA).

[8] Eleftherios Avramidis, Marta R. Costa-jussà, Christian Federmann, Josef van Genabith, Maite Melero, and Pavel Pecina. A Richly Annotated, Multilingual Parallel Corpus for Hybrid Machine Translation. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2189–2193, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

[9] Pau Baquero-Arnal, Javier Jorge, Adrià Giménez, Joan Albert Silvestre-Cerdà, Javier Iranzo-Sánchez, Albert Sanchis, Jorge Civera, and Alfons Juan. Improved Hybrid Streaming ASR with Transformer Language Models. In *Proc. Interspeech 2020*, pages 2127–2131, 2020.

[10] G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.

[11] Yoshua Bengio and Samy Bengio. Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 400–406, Cambridge, MA, USA, 1999. MIT Press.

[12] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003.

[13] Chris Callison-Burch, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia, editors. *Proceedings of the Seventh Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Montréal, Canada, June 2012.

[14] Mauro Cettolo, Christian Girardi, and Marcello Federico. WIT3: Web Inventory of Transcribed and Translated Talks. In *Proceedings of the 16th Annual conference of the European Association for Machine Translation*, pages 261–268, Trento, Italy, May 28–30 2012. European Association for Machine Translation.

[15] Thomas Cherian, Akshay Badola, and Vineet Padmanabhan. Multi-cell LSTM Based Neural Language Model, 2018.

[16] Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Ekaterina Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art Speech Recognition With Sequence-to-Sequence Models, 2017.

[17] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-Based Models for Speech Recognition. In *NIPS*, 2015.

[18] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. In *Machine Learning*, pages 273–297, 1995.

[19] M. A. del Agua, A. Giménez, N. Serrano, J. Andrés-Ferrer, J. Civera, A. Sanchis, and A. Juan. The Translectures-UPV Toolkit. In Juan Luis Navarro Mesa, Alfonso Ortega, António Teixeira, Eduardo Hernández Pérez, Pedro Quintana Morales, Antonio Ravelo García, Iván Guerra Moreno, and Doroteo T. Toledano, editors, *Advances in Speech and Language Technologies for Iberian Languages*, pages 269–278, Cham, 2014. Springer International Publishing.

[20] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805, 2018.

[22] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001.

[23] Mark JF Gales. Maximum likelihood linear transformations for HMM-based speech recognition. *Computer speech & language*, 12(2):75–98, 1998.

[24] Ismael García Varea. *Traducción automática estadística: modelos de traducción basados en máxima entropía y algoritmos de búsqueda*. PhD thesis, Universitat Politècnica de València, 2003. Advisor: Francisco Casacuberta Nolla.

[25] Adrià Giménez Pastor. *Bernoulli HMMs for Handwritten Text Recognition*. PhD thesis, Universitat Politècnica de València, 2014. Advisors: Alfons Juan Ciscar and Jesús Andrés Ferrer.

[26] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5):602 – 610, 2005. IJCNN 2005.

[27] Mokhtar M Hasan and Pramod K Mishra. Robust gesture recognition using gaussian distribution for features fitting. *International Journal of Machine Learning and Computing*, 2(3):266, 2012.

[28] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[29] Kenneth Heafield. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

[30] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[32] Javier Jorge, Adrià Giménez, Pau Baquero-Arnal, Javier Iranzo-Sánchez, Alejandro Pérez, Gonçal V. Garcés Díaz-Munío, Joan Albert Silvestre-Cerdà, Jorge Civera, Albert Sanchis, and Alfons Juan. MLLP-VRAIN Spanish ASR Systems for the Albayzin-RTVE 2020 Speech-To-Text Challenge. In *Proc. IberSPEECH 2021*, pages 118–122, 2021.

[33] Javier Jorge, Adrià Giménez, Javier Iranzo-Sánchez, Jorge Civera, Albert Sanchis, and Alfons Juan. Real-Time One-Pass Decoder for Speech Recognition Using LSTM Language Models. In *Proc. Interspeech 2019*, pages 3820–3824, 2019.

[34] Javier Jorge, Adrià Martínez-Villaronga, Pavel Golik, Adrià Giménez, Joan Albert Silvestre-Cerdà, Patrick Doetsch, Vicent Andreu Císcar, Hermann Ney, Alfons Juan, and Albert Sanchis. MLLP-UPV and RWTH Aachen Spanish ASR Systems for the IberSpeech-RTVE 2018 Speech-to-Text Transcription Challenge. In *Proc. IberSPEECH 2018*, pages 257–261, 2018.

[35] A. Juan and E. Vidal. Bernoulli mixture models for binary images. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 367–370 Vol.3, 2004.

[36] Philipp Koehn. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand, 2005. AAMT, AAMT.

[37] Tin-Yau Kwok and Dit-Yan Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Transactions on Neural Networks*, 8(3):630–645, 1997.

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[39] Johannes Lederer. Activation Functions in Artificial Neural Networks: A Systematic Overview, 2021.

[40] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[41] Geoffrey J. McLachlan and Suren Rathnayake. On the number of components in a Gaussian mixture model. *WIREs Data Mining and Knowledge Discovery*, 4(5):341–355, 2014.

[42] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Coherent Dialogue with Attention-Based Language Models. In *AAAI*, 2017.

[43] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. volume 2, pages 1045–1048, Jan 2010.

[44] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[45] David Nolden. *Progress in Decoding for Large Vocabulary Continuous Speech Recognition*. PhD thesis, RWTH Aachen University, Computer Science Department, RWTH Aachen University, Aachen, Germany, April 2017.

[46] Mohammadreza Asghari Oskoei and Huosheng Hu. Support Vector Machine-Based Classification Scheme for Myoelectric Control Applied to Upper Limb. *IEEE Transactions on Biomedical Engineering*, 55(8):1956–1965, 2008.

[47] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. Fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[48] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks, 2012.

[49] L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[50] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation, 2021.

[51] Anupama Ray, Sai Rajeswar, and Santanu Chaudhury. Text recognition using deep BLSTM networks. In *2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR)*, pages 1–6, 2015.

[52] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[53] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[54] Haşim Sak, Andrew Senior, Kanishka Rao, and Françoise Beaufays. Fast and Accurate Recurrent Neural Network Acoustic Models for Speech Recognition, 2015.

[55] Alex Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306, mar 2020.

[56] Yongzhe Shi, Wei-Qiang Zhang, Meng Cai, and Jia Liu. Efficient One-Pass Decoding with NNLM for Speech Recognition. *IEEE Signal Processing Letters*, 21(4):377–381, 2014.

[57] Joan Albert Silvestre-Cerdà, Miguel Del Agua, Gonçal Garcés, Guillem Gascó, Adrià Giménez-Pastor, Adrià Martínez, Alejandro Pérez González de Martos, Isaías Sánchez, Nicolás Serrano Martínez-Santos, Rachel Spencer, Juan Daniel Valor Miró,

Jesús Andrés-Ferrer, Jorge Civera, Alberto Sanchís, and Alfons Juan. transLectures. In *Proceedings of IberSPEECH 2012*, pages 345–351, Madrid (Spain), 2012.

[58] Ralf Steinberger, Mohamed Ebrahim, Alexandros Poulis, Manuel Carrasco-Benitez, Patrick Schlüter, Marek Przybyszewski, and Signe Gilbro. An overview of the European Union's highly multilingual parallel corpora. *Language Resources and Evaluation*, 48(4):679–707, 2014.

[59] G. Stemmer, F. Brugnara, and D. Giuliani. Adaptive training using simple target models [speech recognition applications]. In *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, volume 1, pages I/997–I1000 Vol. 1, 2005.

[60] Andreas Stolcke. SRILM – An extensible language modeling toolkit. In *IN PROCEEDINGS OF THE 7TH INTERNATIONAL CONFERENCE ON SPOKEN LANGUAGE PROCESSING (ICSLP 2002)*, pages 901–904, 2002.

[61] Pawel Swietojanski, Arnab Ghoshal, and Steve Renals. Revisiting hybrid and GMM-HMM system combination techniques. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6744–6748, 2013.

[62] Jörg Tiedemann. News from OPUS — A collection of multilingual parallel corpora with tools and interfaces. 2009.

[63] Vidushi Vashishth, Anshuman Chhabra, and Deepak Kumar Sharma. GMMR: A Gaussian mixture model based unsupervised machine learning approach for optimal routing in opportunistic IoT networks. *Computer Communications*, 134:138–148, 2019.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. *ArXiv*, abs/1706.03762, 2017.

[65] Voxforge.org. Free Speech... Recognition (Linux, Windows and Mac). http://www.voxforge.org/. Accessed on 17/06/2022.

[66] Chenguang Wang, Mu Li, and Alexander J. Smola. Language Models with Transformers. *CoRR*, abs/1904.09408, 2019.

[67] Y. Wang, A. Mohamed, D. Le, C. Liu, A. Xiao, J. Mahadeokar, H. Huang, A. Tjandra, X. Zhang, F. Zhang, C. Fuegen, G. Zweig, and M. L. Seltzer. Transformer-Based Acoustic Modeling for Hybrid Speech Recognition. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6874–6878, 2020.

APPENDIX

SUSTAINABLE DEVELOPMENT GOALS

Degree to which the work relates to the Sustainable Development Goals (SDGs).

Sustainable development goals	**High**	**Medium**	**Low**	**Not applicable**
SDG 1. **No poverty.**				**X**
SDG 2. **Zero hunger.**				**X**
SDG 3. **Good health and well-being.**			**X**	
SDG 4. **Quality education.**	**X**			
SDG 5. **Gender equality.**				**X**
SDG 6. **Clean water and sanitation.**				**X**
SDG 7. **Affordable and clean energy.**				**X**
SDG 8. **Decent work and economic growth.**			**X**	
SDG 9. **Industry, innovation and Infrastructure.**		**X**		
SDG 10. **Reduced Inequality.**		**X**		
SDG 11. **Sustainable cities and communities.**				**X**
SDG 12. **Responsible consumption and production.**				**X**
SDG 13. **Climate action.**				**X**
SDG 14. **Life below water.**				**X**
SDG 15. **Life on land.**				**X**
SDG 16. **Peace and justice strong institutions.**			**X**	
SDG 17. **Partnerships to achieve the goal.**			**X**	

Reflexion on the relation of the TFG/TFM with the SDGs and with the most related SDG(s).

This work fits with the United Nations' Sustainable Development Goals (SDGs). In particular, with SDG 4, on "Quality Education", which aims to "ensure inclusive and equitable quality education and promote lifelong learning opportunities for all".
Within SDG 4, the most related targets to this work are:

- *4.3 By 2030, ensure equal access for all women and men to affordable and quality technical, vocational and tertiary education, including university.*
- *4.4 By 2030, substantially increase the number of youth and adults who have relevant skills, including technical and vocational skills, for employment, decent jobs and entrepreneurship.*
- *4.5 By 2030, eliminate gender disparities in education and ensure equal access to all levels of education and vocational training for the vulnerable, including persons with disabilities, indigenous peoples and children in vulnerable situations.*

This work contributes to increasing accessibility to educational resources for everyone (target 4.3), including people with hearing disabilities and persons with fewer resources or without access to formal education systems (target 4.5), aside from those whose acces to educational and formation resources requires them to learn and master a foreign language (target 4.4).