Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Development and evaluation of a Polish Automatic Speech Recognition system using the TLK toolkit

DEGREE FINAL WORK

Degree in Computer Science

Author: Nahuel Unai Roselló Beneitez

Tutor: Albert Sanchis Navarro
Cotutor: Jorge Civera Saiz
Experimental director: Javier Iranzo Sánchez

Course 2019-2020

Resum

El reconeixement automàtic de la parla (ASR per les seues sigles en anglés) és una branca del reconeixement de patrons que ha guanyat popularitat gràcies als avanços tecnològics i estructurals en aquest camp. Aquest treball presenta el procés de creació d'un sistema automàtic reconeixedor de la parla polonesa basat en tècniques actualment estat de la tècnica. Amb aquesta finalitat, s'utilitza l'eina TLK, a més d'altres eines rellevants. D'altra banda, s'examinen i es descriuen els conjunts de dades usats per a entrenar els models finals. A més, el procés de creació dels models es descriu partint des del principi. Finalment, el sistema desenvolupat es compara amb altres sistemes avaluats en el context de la competició de PolEval 2019.

Paraules clau: reconeixement de formes, aprenentatge automàtic, reconeixement automàtic de la parla, aprenentatge profund, TLK, polonés

Resumen

El reconocimiento automático del habla (ASR por sus siglas en inglés) es una rama del reconocimiento de patrones que ha ganado una gran popularidad en los últimos años gracias a los avances tecnológicos y estructurales en este campo. Este trabajo presenta el proceso de creación de un sistema de reconocimiento automático del habla polaca basado en técnicas actualmente estado de la técnica. Con este fin, se hace uso de la herramienta TLK, además de otras herramientas relevantes. También se examinan y describen los conjuntos de datos usados para entrenar los modelos finales, además de describirse el proceso de creación de estos últimos partiendo desde el principio. Finalmente, el sistema desarrollado se compara con otros sistemas evaluados en el contexto de la competición de PolEval 2019.

Palabras clave: reconocimiento de formas, aprendizaje automático, reconocimiento automático del habla, aprendizaje profundo, TLK, polaco

Abstract

Automatic speech recognition (ASR) is a branch of pattern recognition that has currently gained popularity due to technological and structural advances in this field. This work presents the process of making a Polish automatic speech recognition system based on current state-of-the-art techniques. To this end, the TLK toolkit is used, as well as other relevant tools. The datasets used to train the final models are examined and described. Moreover, the process of creating the models from scratch is described. To conclude, the performance of the final system is compared with respect to several systems evaluated in the context of the PolEval 2019 challenge.

Key words: pattern recognition, machine learning, automatic speech recognition, deep learning, TLK, Polish

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This work studies the training and evaluation of an automatic speech recognition system. To achieve this, several resources from the Polish language and specialized toolkits such as TLK, as well as other relevant software, will be used.

This chapter discusses the motivation of this work. Moreover, the goals of this work are described, and the document structure is outlined.

1.1 Motivation

The main communication tool for humanity is the language. Understanding and processing another person's voice is not an easy task. However, as humans, we are able to do it because of our rationality and intelligence. The field of automatic speech recognition (ASR) aims to pass down such knowledge to machines, and because of this, it has been one of the most highlighted fields of pattern recognition for years.

Training a state-of-the-art ASR system which can recognise any language is a perfect introduction to this highly demanded field. Moreover, this first contact establishes the basis to understand current ASR practices which are considered state-of-the-art.

The Polish language is spoken by more than 50 million people worldwide, so it should not be considered a minority language. Although there are not as many Polish resources as in other languages such as English or Spanish, there is more than enough data available online to train an ASR system. Projects such as Mozilla Common Voice[1] are compiling available audio on the Internet and creating more audio from voluntary speakers in order to build several voice datasets, including a specific dataset for the Polish language.

Despite this, there are few reports of state-of-the-art ASR systems oriented to the Polish language. In [30, 31], systems similar to the one developed in this work are described. Moreover, there have been efforts in order to develop state-of-the-art architectures which model the Polish language [2, 13, 43][2]. However, to the author's best knowledge, no research has been made yet in the combination of these current state-of-the-art language models with ASR, which are supposed to benefit the resulting system.

[1]Mozilla Common Voice is a collaborative attempt from the Mozilla Foundation to build voice datasets for several languages in order to ”help make voice recognition open and accessible for everyone”: https://voice.mozilla.org/en. Unfortunately, the Polish language was not available at the time of gathering the resources.

[2]https://github.com/kldarek/polbert. Last accessed: July 1st, 2020.

1.2 Main Objectives

The main goals of this work are the following:

- To apply general concepts and state-of-the-art techniques involved in training ASR systems for real-life applications.
- To interpret the underlying theoretical concepts of ASR systems.
- To use the TLK toolkit as well as other tools required in order to develop and train a state-of-the-art ASR system.
- To evaluate the performance of the developed system.
- To compare the performance of the developed system with respect to competitive Polish ASR systems.

1.3 Document Structure

This document is divided into six main chapters:

- Chapter 1, the current one, has established a motivation in order to create an ASR system.
- Chapter 2 introduces the reader to the field of pattern recognition and gives an in-depth perspective of ASR while justifying some of the key choices made in this domain in the past.
- Chapter 3 details the datasets used in order to train both the language and the acoustic models.
- Chapter 4 details the work performed on the acoustic model training.
- Chapter 5 describes the language model training, as well as the text data preprocessing.
- Chapter 6 evaluates the final developed system and compares it with other similar systems.
- Chapter 7 ends this work with the conclusions extracted from the tasks performed, the lessons learned from completing this work, and future work that may be done regarding the developed system.

CHAPTER 2

Fundamentals of Automatic Speech Recognition

2.1 Introduction

ASR could be defined as the process which allows obtaining the text uttered from a given speech, represented as an audio signal. It is a cheap and effective way of transcribing audio into text: once trained, ASR systems can keep transcribing audio into text indefinitely, or until further improvements enter the field and the system is updated.

Recent figures and comparisons on both human versus ASR performance and current versus past ASR systems show a significant decrease in error rate. For instance, Lippmann [28] establishes in 1997 a minimum achieved of 43% for the Switchboard dataset[1], whereas a 2017 paper from Saon and Kurata [45] sets the error rate of such dataset at 5.1%. Moreover, a recent article sets a positive correlation between human and computer understanding of utterances in this dataset, with results that are "qualitatively surprisingly similar to professional human transcriber output" [52].

The field of ASR has recently gained a lot of traction due to smart gadgets and the *Internet of Things*, whose objective is to interconnect daily life items. Paired with natural language processing, a machine can not only understand the uttered words but also try to infer the meaning of the message provided by the speaker: virtual assistants such as Google Assistant[2], Apple's Siri[3], Microsoft's Cortana[4] or Amazon's Alexa[5] make significant use of ASR to understand the user and comply with their orders.

Combined with machine translation, ASR can take audio from a speaker and translate it into a different language. Furthermore, ASR can also be used on its own, for instance in video subtitling. As can be imagined, there are many applications of ASR, and plenty of systems that use this kind of technology nowadays.

ASR can also be applied to other fields such as the medical field. It can help in speech disability recognition and treatment: for instance, in the task of understanding people with disabilities such as dysarthria, a type of motor disability that prevents coordination of the speech musculature which leads to imprecise articulation, decreased rate of speech,

[1]The Switchboard dataset is a collection of telephone conversations with approximately 260 hours of English speech: `https://catalog.ldc.upenn.edu/LDC97S62`.

[2]`https://assistant.google.com/`.

[3]`https://apple.com/siri/`.

[4]`https://microsoft.com/en-us/cortana`.

[5]`https://developer.amazon.com/en-GB/alexa`.

and reduced intelligibility[6]. Even if humans are still better at this task than ASR systems, a study from Mengistu and Rudzicz claims that the latter can reach a 68.39% accuracy, whereas the human counterpart resides at a 79.78% [33].

Vocaliza [53] is an application aimed at supporting speech therapists in the rehabilitation of Spanish speakers that suffer speech disorders. Those readers interested in the field of speech disorder treatment through ASR are pointed to [36].

2.2 Pattern Recognition

Pattern recognition is a widely used approach to recognizing and classifying samples according to criteria (or patterns) which are common to all of them. According to Bishop [10, p. 1], "the field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories".

Before explaining deeper terminology, some core concepts will be briefly described:

- The concept of *class*: it represents a group of data that share similar features. For instance, a classical example of pattern recognition distinguishes between two opposite classes that an email can belong to: *spam* and *no-spam*, meaning that either it can contain spam or it is thought to be a safe email.

- The concept of *object*: in pattern recognition, the term *object* usually refers to a vector $\mathbf{x} \in \mathbb{R}^D$, product of extracting features from a real-life *sample*, with each dimension D being a relevant value of that which we want to measure. Another classic example consists of distinguishing flowers from three classes: *setosa*, *versicolor* and *virginica*. Some commonly used dimensions for this problem are the width and length of both the petals and the sepals, yielding a total of four dimensions in this example.

- The concept of *classification*: it has been implicitly introduced in the former points. *Classifying* an object into a class means choosing the most suitable class for an object according to a decision rule. This decision rule is obtained by applying a given function $f\colon \mathbf{x} \to C$, where $\mathbf{x}$ is the object to be classified and C is the set of classes of the problem.

As mentioned when describing the concept of object, it is usually convenient to take the most relevant features of the sample. The task of mapping this gathered information into a set of objects or feature vectors is called preprocessing, being an essential step in nearly every pattern recognition application. In the flower classification example previously explained, preprocessing could simply mean measuring both the petals and the sepals, and then placing such data in a vector. The task of preprocessing an acoustic signal in ASR is described in Section 2.4.1.

The classification function plays a critical role in the context of pattern recognition. It is most commonly associated to Bayes' theorem, which relates the probability of an object $\mathbf{x}$ belonging to a class c to the probability of the class yielding the given object:

$$P(c \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c)\, P(c)}{P(\mathbf{x})} \tag{2.1}$$

[6]Pamela Enderby. Dysarthria. In *Handbook of Neurological Rehabilitation*, pages 357–362. Psychology Press, 2003.

The previous equation may be used in order to classify an object to a certain class. This estimated class $\hat{c}$ will then be the one which yields the maximum value of $P(c \mid \mathbf{x})$:

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c \mid \mathbf{x}) = \operatorname*{argmax}_{c \in C} \frac{P(\mathbf{x} \mid c)\, P(c)}{P(\mathbf{x})} \tag{2.2}$$

Two things must be noted when looking at the previous equation. The first one is that maximizing the given function means minimizing the error associated with such function. The second one is that the probability of the object $\mathbf{x}$ is the same no matter what the class is, since $P(\mathbf{x})$ does not depend on the class of the object but on the object itself. Therefore, the denominator does not play any role in maximizing the function for any specific class so it can be safely dropped:

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(\mathbf{x} \mid c)\, P(c) \tag{2.3}$$

The main problem when developing a pattern recognition system lies in the fact that estimating those parameters is generally very complex, and in practically all non-trivial problems the system must rely on approximations to these probabilities. Pattern recognition aims to create a system that classifies the maximum number of objects in the class to which they belong by tuning these parameters.

In the case of supervised learning, a set of labeled data (objects with their respective classes) is used to train the system, being the main objective to classify unlabeled data (objects of which we do not know the class) in the class that each object $\mathbf{x}$ is most likely to belong to. Figure 2.1 shows the basic structure of a pattern recognition system.

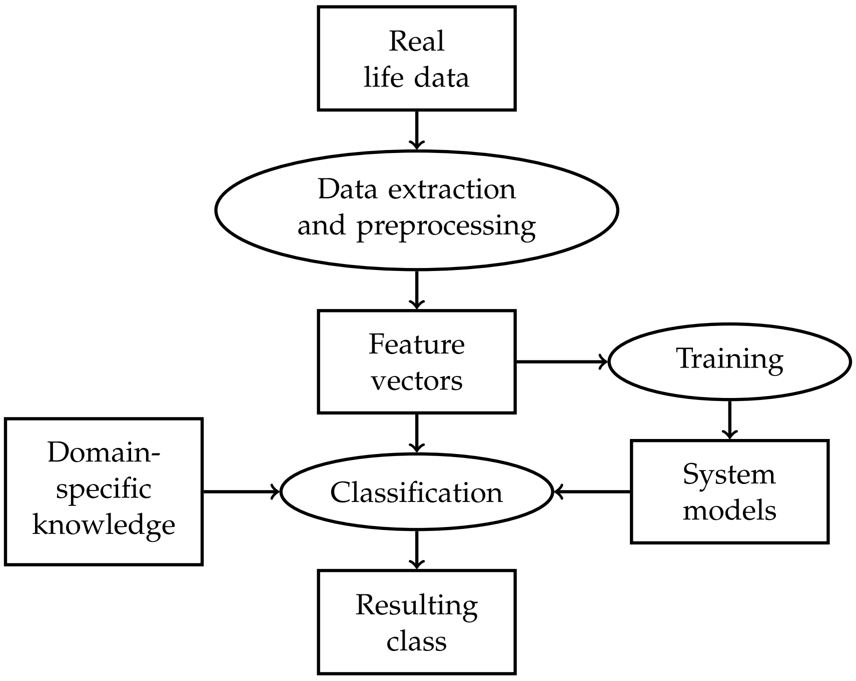


Figure 2.1: Design of the basic structure of a pattern recognition system. Circles design actions, while squares reference available data or information at each step.

2.3 Neural Networks

Before going further in the work, it is vital to know about neural networks since they will be mentioned many times throughout this work. A neural network, also sometimes called connectionist model, is a set of simple processors (neurons) which form connections among them.

Neural networks are distributed in three layers. The *input layer* receives the input and passes it forward to the *hidden layer*, which in turn sends it to the *output layer*, that yields the output. In both the hidden and the output layers, computations are performed according to the following formula:

$$s_i^j(\mathbf{x}, \boldsymbol{\theta}) = g(\sum_{k=0}^{M_{j-1}} \theta_{ik}^j \, x_k) \tag{2.4}$$

where j is the layer index, i is the neuron index within the layer, M_l is the number of neurons at layer l, $\mathbf{x}$ is the input received from the previous layer, $\boldsymbol{\theta}$ is the set of parameters (more commonly called *weights*) which must be learned, and $g(\cdot)$ is an *activation function*, usually non-linear, which scales the neuron's output to an admissible range. For instance, a common activation function is the softmax function. This function is applied over a set of values $K = \{k_1, k_2, \ldots, k_n\}$, yielding a probability distribution among them:

$$g(k_i) = \frac{\exp(k_i)}{\sum_{j=1}^{n} \exp(k_j)} \tag{2.5}$$

where $\exp(x)$ is the exponential function e^x.

Neural networks are used in classification problems due to their discriminative capabilities. In essence, they can learn to classify objects into classes by being provided labeled examples when being trained. In these cases, the output should be a probability, which can be achieved by using the softmax activation function on the output layer.

The set of weights $\boldsymbol{\theta}$ can be learned through the backpropagation algorithm, which works in two phases: first, the corresponding values at each neuron are computed by processing the input through Equation 2.4. When the output has been computed, the error of the network is calculated by comparing the expected output with the actual output and propagating the error backwards (hence the name). Every weight is then scaled by the error obtained from the following layer. For more details and a formal definition of this algorithm, the reader is pointed to [42].

Some readers may have noticed that Equation 2.4 is very similar to the perceptron formula [41]. Neural networks are in fact also called multilayer perceptrons: every neuron computes a single discriminative function according to the set of weights that reach such neuron, and the set of discriminative functions computed by the neurons allows discerning a class from the rest, making it possible to learn complex patterns. Figure 2.2 shows a generic neural network.

2.3.1. Advanced Neural Network Models

It has been proven by Huang and Lippmann that a neural network can learn any arbitrary function given enough neurons. However, this comes at the cost of computational time [21]. By this statement, it may seem irrelevant that a neural network with more than a single hidden layer can benefit accuracy. However, neural networks with multiple hidden layers, or deep neural networks (DNNs), have been found to approximate functions

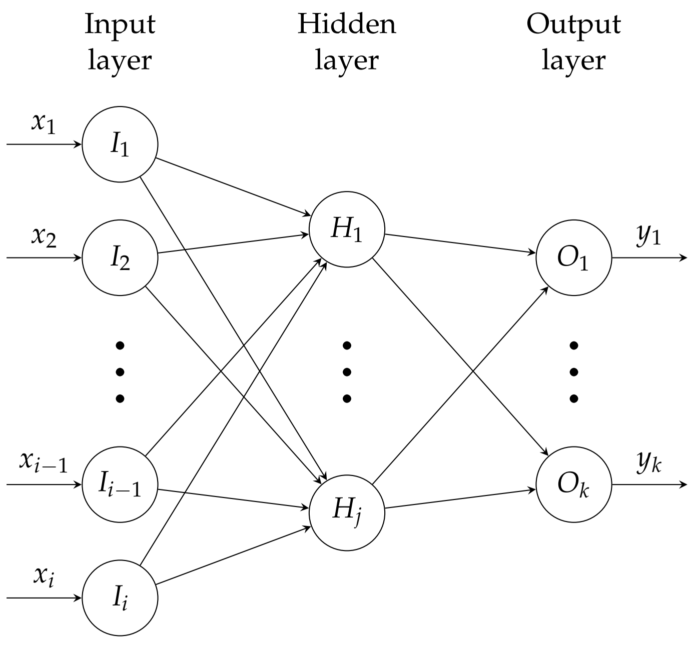


Figure 2.2: General structure of a neural network. In this neural network, $M_0 = i$, $M_1 = j$ and $M_2 = k$. Each neuron m is connected with every neuron n of the following layer n by a weight, which is usually represented as θ_{nm}.

in the same way than regular neural networks by using an exponentially lower number of neurons [27].

It would be beneficial for this kind of networks, however, to carry information about the past. Recurrent neural networks (RNNs) are a type of networks that solve this problem by having their units connected in such a way that cycles appear. These cycles trap past information and are associated with a time-delay operation [59].

Bidirectional long short-term memory (BLSTM) networks are a subset of RNNs which try to improve upon the basic RNN structure. The LSTM architecture aims to implement memory in specific cells due to the fact that, at a given time, the error propagated through a network either vanishes or grows exponentially and, as a consequence, weights may either oscillate or not change at all, which is known as the exploding or vanishing gradient phenomenon [20].

The LSTM architecture implements memory by introducing the concepts of memory cell, which retains information, and gate units. There are three main types of gate units in a memory cell: input gates are in charge of controlling error flow arriving at the memory cell, output gates control the error flow out of the memory cell, and forget gates control whether the contents of the LSTM cell disappear or not. Input gates can be seen as the ones that decide whether to keep or override information in the cell coming from outside the cell, whereas output gates are in charge of letting (or preventing) information flow out of the cell [20]. Figure 2.3 shows the basic scheme of a memory cell.

Bidirectionality is achieved by having both past and future information about the input data. This is achieved by splitting the basic unit of the network into two parts: "a part that is responsible for the positive time direction (forward states), and a part for the negative time direction (backward states)", making it possible to train the network using the general algorithm to train an RNN, the backpropagation through time (BPTT) algorithm [46]. Figure 2.4 shows an example of a bidirectional network.

Recently, the focus of investigation has been shifted towards an attention-based mechanism. This kind of mechanisms "iteratively process their input by selecting relevant

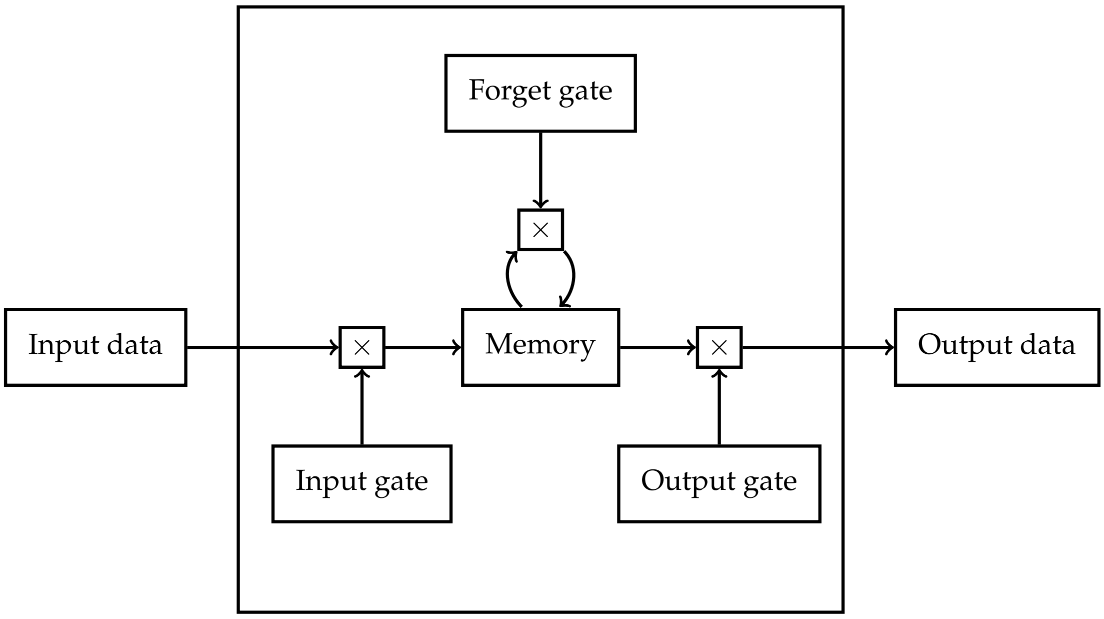


Figure 2.3: Basic schema of an LSTM memory cell. Nodes labeled with × denote the multiplication operation performed on the inputs that arrive to such node.

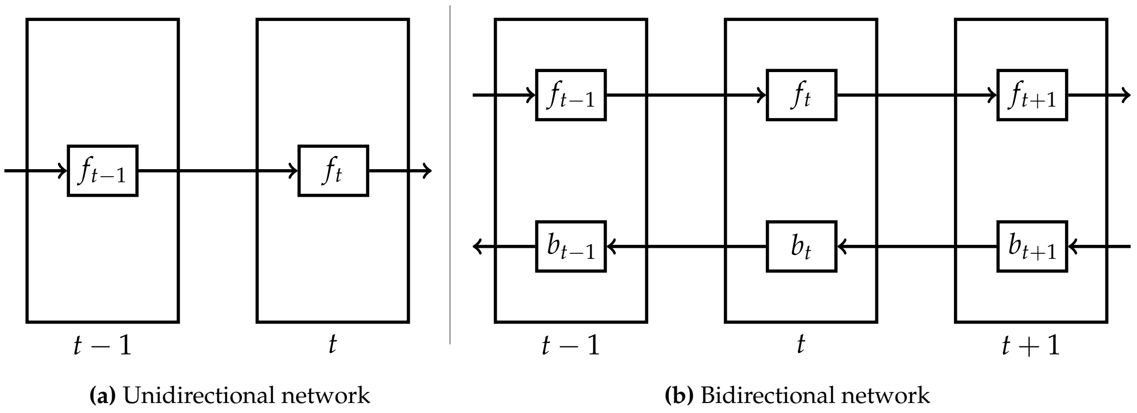


Figure 2.4: Unidirectional network (left) versus bidirectional network (right). These images correspond to an unfolded recurrent neural network, where loops are substituted to edges pointing to the same unit in a different time instant t. In a bidirectional network, a unit f_i is responsible for the communication in the positive time direction (starting from the beginning of the sample), while another unit b_j communicates in the negative time direction (starting from the end of the sample). These are known as the *forward states* and the *backward states*, respectively.

content at every step" by focusing on the most relevant parts of what needs to be processed [12]. Attention-based recurrent networks encode information by bidirectionally analyzing the input sequence and obtaining an attention vector for each word in the sequence [3].

Attention, however, needs a processing time of $O(n)$, or $O(\log n)$ in carefully developed systems, being n the size of the input to be processed. The transformer [54] is an architecture which reduces the time taken when relating two arbitrary signals from the input or the output (in essence, computing attention) to $O(1)$ since it avoids recurrence-based mechanisms. Another consequence of this recurrency avoidance is the parallelization that this architecture brings.

The transformer is composed of layers. Each layer is in turn composed of sublayers that compute attention over the received inputs and contain fully-connected neural networks [54]. Since its proposal, the transformer architecture has experienced enhancements, for instance with the addition of bidirectionality [16] or LSTM layers [56]. For a deeper understanding of the transformer architecture, the reader is pointed to [24].

2.4 Automatic Speech Recognition

ASR refers to the capability of a system to recognize the user's voice and represent it as text. More formally, it tries to assert the most probable sequence of words (sentence) given a sequence of acoustic vectors that have been obtained by analyzing the utterance. This fits in the definition of pattern recognition by taking the sequence of acoustic vectors a as the object $\mathbf{x}$, and the sentence w as the class c. With this in mind, Equation 2.3 can be remodeled as follows to fit the needs of ASR:

$$\hat{w} = \underset{w \in L}{\operatorname{argmax}} P(w \mid a) = \underset{w \in L}{\operatorname{argmax}} P(a \mid w)\, P(w) \tag{2.6}$$

where L is the set of possible sentences from the language. In the previous equation, $P(w)$ denotes the probability that the sequence of words w appears in a sentence while $P(a \mid w)$ is the probability that such sequence of words generates the sequence of acoustic vectors a. These factors can be calculated with a language model and an acoustic model, respectively.

2.4.1. Acoustic Data Preprocessing

It is useful for the statistical classifiers to work with data whose number of dimensions is fixed. The technique of taking an acoustic signal and creating vectors that hold the most relevant features is called preprocessing, and it is a crucial step in most of the tasks related to machine learning, as already seen in Figure 2.1. Audio preprocessing consists of several steps that are explained in the following points[7]:

1. The audio goes first through a pre-emphasis stage. This step tries to lower the difference between lower and higher frequencies by applying a formula similar to a derivative in order to synthesize the most relevant changes in the audio and reducing some of the recorded utterance's noise. The resulting audio x' may be calculated with the following formula:

   $$x'(n) = x(n) - k \cdot x(n-1) \tag{2.7}$$

   where x is the raw audio and k is a constant $0 \leq k < 1$, usually equal to 0.97.

   Pre-emphasis used to show an improvement in the past. However, it has not shown a significant accuracy increase in combination with present ASR practices, and therefore it has been dismissed in current, newer systems.

2. When preprocessing audio to extract the most relevant features, not all of the audio is processed at once. Instead, feature vectors are extracted from time windows. In order to get these time windows, two parameters must be set: the window size, which defines the block size of the audio to be preprocessed, and the period, that establishes the gap between a window and its adjacent one. Both parameters are chosen so that the window size is greater than the period and are in the order of milliseconds.

[7]To avoid confusion when dealing with multiplications, functions and other mathematical terms, the $\cdot$ operator will be explicitly used in this section whenever a multiplication operation is declared (except in very obvious cases such as 2π).

The application of the time windows to the audio yields a set of overlapping windows in which, due to its reduced size, it can be assumed that only a single phoneme is pronounced. At first, overlapping time windows may seem to carry redundant information. Nevertheless, according to Frühholz and Belin [17, pp. 709-710], it is needed due to the fact that information could be missed or lost if windows were non-overlapping and because the exact phoneme location is unknown. Figure 2.5 shows a graphical representation of the window size and the period.

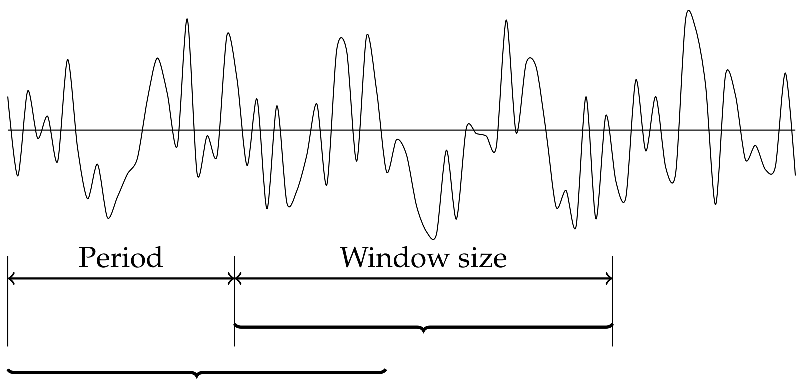


Figure 2.5: Graphical representation of the window size and period parameters when extracting windows from an audio signal. The values that TLK uses are 25 ms for the window size and 10 ms for the period, yielding a total of approximately 100 windows per second.

3. The Fourier transform is applied to the resulting time windows. This operation translates every frame from the temporal domain into the frequential domain by taking all of the amplitudes that compose the wave located at the considered time window. The basic equation of the Fourier transform is defined as follows:

$$X_i(l) = \sum_{n=0}^{L-1} x_i(n) \cdot \exp(-j \cdot n \cdot l \cdot \frac{2\pi}{N}) \quad \forall\, l \in \{0, 1, \ldots, L-1\} \tag{2.8}$$

   where L is the size of the desired vector, usually equal to 512 when the sampling frequency is 16 kHz.

4. Mel filter banks are applied to the frequential data obtained through the previous step. These filter banks try to simulate the fact that the human does not hear every frequency equally: while it is desired to thoroughly recognize lower frequencies, higher tones are assumed to be less discriminative. The number of dimensions of the final vector is equal to the number of filters that compose the filter bank.

   The value of the function at a given point of each filter can be then calculated as [22, p. 314]:

$$H_m(k) = \begin{cases} 0 & \text{if } k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & \text{if } f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & \text{if } f(m) \le k \le f(m+1) \\ 0 & \text{if } k > f(m) \end{cases} \tag{2.9}$$

   where m and $f(m)$ are the m-th filter and the peak value of the m-th filter, respectively[8]. Through the use of the previous equation, the final vector of size M can be

[8]It is assumed that $f(0)$ is the value at the leftmost point of the first filter's base and $f(M+1)$ is the value at the rightmost point of the last filter's base.

calculated:

$$m_i = \sum_{l=0}^{\frac{L}{2}-1} X_i(l) \cdot H_m\left(F_m \cdot \frac{l}{L}\right) \tag{2.10}$$

where F_m is the sampling frequency. This equation scales the output of the Fourier transform $X_i(l)$ by the value of each filter H_m, and then it sums all these values in order to get one dimension of the final vector; only the first $\frac{L}{2}$ components are used due to the symmetry of the Fourier transform. Figure 2.6 shows a Mel filter bank with 16 filters.

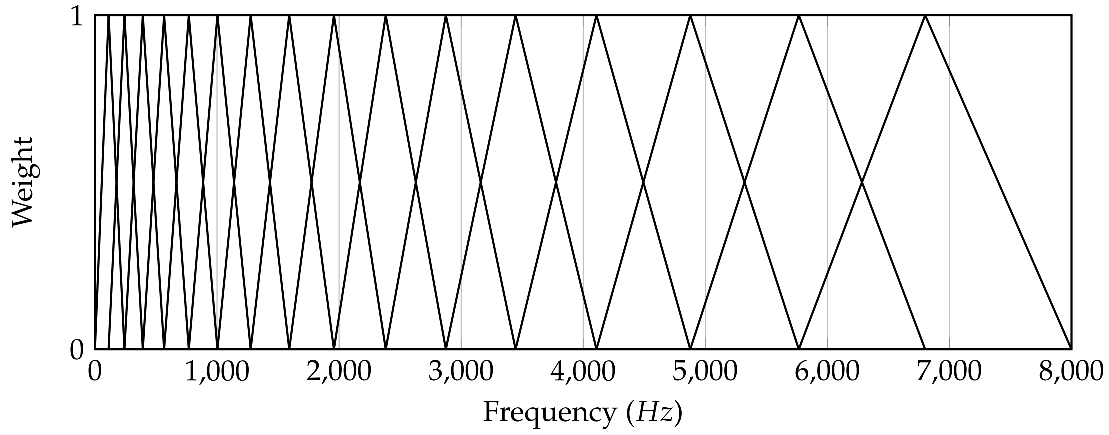


Figure 2.6: Mel filter bank consisting of 16 different filters. Each of the different filters corresponds to a triangle.

5. Each of the components is scaled logarithmically in order to model the human's non-linear sensitivity scale. Then, the discrete cosine transform (DCT) is used. This operation decorrelates the filter bank coefficients, and therefore the final vector elements. The following equation corresponds to the application of the DCT to a vector of size K:

$$c_i(j) = \sum_{k=1}^{K} m_i(k) \cdot \cos\left(\frac{\pi j}{K} \cdot (k - 0.5)\right) \tag{2.11}$$

6. The final step consists of normalizing the sample, which aims at minimizing differences between audios. As a result, the vector obtained has a mean value $\mu = 0$ and a variance $\sigma = 1$. If this step was not performed, irrelevant differences with regards to phoneme pronunciation such as loudness could play an important role in obtaining the transcription of the utterance.

As a result of these transformations, a normalized *Mel Frequency Cepstral Coefficient* (MFCC) vector is obtained for each frame. These vectors are passed as an input to the acoustic model, which will then return the most probable phoneme associated to every feature vector, as seen in subsection 2.4.2.

2.4.2. Acoustic Model

The acoustic model is the part of the system which is in charge of analyzing the audio and outputting the most probable phonemes for a specific window of the audio. Well-known classifier techniques such as support vector machines (SVM) or artificial neural networks (ANN) are not straightforward to use in this task since they have to be applied on a fixed-size input, whereas speech signal can be indefinitely extended through time.

Even if audio preprocessing can solve this problem by creating fixed-size vectors from the utterance, there is still an unknown number of vectors. This calls for structures that can correctly manage temporal flow and transition from a given state to another one only when needed. Hidden Markov models (HMMs) can perform this task, and consequently, they have been widely used for speech recognition.

HMMs oriented to speech recognition aim to model phonemes (or triphonemes; more details on Section 4.4) and silence. They follow a strictly linear structure[9] and are usually composed of three states. The statistical foundations of the utility of HMMs for the recognition of a sentence w lies in the following formula:

$$P_w(a) = \sum_{s \in S} P(a,s) = \sum_{s \in S} P(s)\ P(a \mid s) \tag{2.12}$$

where S is the set composed by every sequence of HMM states s such that $|s| = |a|$. This last restriction is applied because a sequence of length $|a|$ can only be generated by an HMM by emitting, and therefore transitioning, a total of $|a|$ times.

HMMs are based on the Markov assumptions [40]:

1. The state at time t only depends on the state at time $t-1$. Moreover, the transition from a state i to another state j is equiprobable for every time t considered.

2. The emission at time t only depends on the state at time t.

By applying these assumptions to the previous formula, it can be further decomposed:

$$P_w(a) = \sum_{s \in S} \prod_{i=1}^{|s|} P(s_i \mid s_{i-1})\ P(a_i \mid s_i) \tag{2.13}$$

where s_i is the i-th state visited in the sequence of states s, a_i is the i-th vector from the vector sequence a, $P(s_i \mid s_{i-1})$ is the transition probability from s_{i-1} to s_i, and $P(a_i \mid s_i)$ is the emission probability of the vector a_i in the state s_i. Note that in this case $P_w(a) \equiv P(a \mid w)$ because every sentence has an associated set of HMMs, product of the concatenation of one or more tri-state HMMs, which implicitly calculates the posterior probability. Figure 2.7 shows the general structure of an HMM oriented to speech recognition.

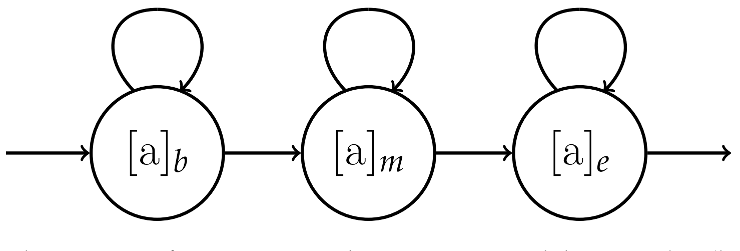


Figure 2.7: General structure of an HMM used in acoustic modeling. In this (hypothetical) case, the phoneme to be recognised is [a], being $[\mathrm{a}]_b$, $[\mathrm{a}]_m$ and $[\mathrm{a}]_e$ the states that model the beginning, the middle and the end of the phoneme, respectively. Only the transition edges are represented.

The emission probability, $P(a_i \mid s_i)$, is estimated through a Gaussian mixture model (GMM) by using the expectation-maximization (EM) algorithm [15].

More recent approaches to speech recognition use DNNs in order to estimate HMM emissions since this approach makes no assumptions about the nature of the data source, which builds a more robust model. In contrast, GMMs assume that data come from a weighted sum of a set of Gaussian distributions, which may not be a solid assumption in

[9]Strictly linear HMMs only have single-node loops and transitions to the immediately next state.

the context of speech recognition due to many unpredictable facts such as environmental noise, which is a factor taken into account by Gaussian distributions when training; this can be reduced by using discriminative training up to a certain degree [48]. Aspects such as the one described lead to a hybrid model, also known as DNN-HMM model.

As it has been mentioned in the previous paragraph, the DNN in the DNN-HMM hybrid model is in charge of estimating the emission probabilities. By applying Bayes' theorem to the emission probabilities defined in Equation 2.13:

$$P(a_i \mid s_i) = \frac{P(a_i)\,P(s_i \mid a_i)}{P(s_i)} \tag{2.14}$$

The discriminative capabilities of the DNN allow computing $P(s_i \mid a_i)$ by learning from examples, while $P(s_i)$ can be calculated by normalizing the counts of every state s_i seen during training. $P(a_i)$ is very difficult to calculate; however, similarly to the transition from Equation 2.2 to Equation 2.3, the fact that the sequence of states with the maximum probability is wanted implies that $P(a_i)$ can be dismissed because it will be the same in all measurements.

RNNs have shown excellent performance in acoustic modeling. In fact, state-of-the-art acoustic modeling involves using BLSTM networks. Past information is useful to model phoneme context, since two identical word sequences may not carry the same acoustic information: for instance, one sentence may be uttered more slowly than the other. Experiments on phoneme recognition show that BLSTMs outperform other types of neural networks, such as LSTMs [18].

Recent work points to the fact that the transformer architecture has become the new state-of-the-art used for acoustic modeling, even in comparison to BLSTM networks: transformer-based models outperform BLSTMs by a minimum of 2% and a maximum of 11% (relative percentages) on two different tasks, as reported by [57].

2.4.3. Language Model

The language model (LM) in ASR aims to represent the structure of a language. In essence, it calculates the probability that a given word appears in combination with others, usually by looking at the word's previous context.

The most commonly used language model is the n-gram model, where the probability of the final word w_n is expressed as the conditional probability $P(w_n \mid w_1^{n-1})$, where $w_1^{n-1} = w_1\, w_2\, \ldots\, w_{n-1}$ is the word's history or context, also expressed as h_n.

The previous expression measures the probability of obtaining the word w_n after having seen the sequence of words w_1^{n-1}. For instance, in a bigram (also known as n-gram of order two or two-gram) language model, the context is limited to a single word. In this case, the probability that the word "world" follows the word "hello" may be expressed as $P(\text{“}world\text{“} \mid \text{“}hello\text{“})$.

These probabilities can be empirically estimated by normalizing the counts of sequence words up to a certain length n. This process should take into account that they should be long enough and representative of the language structure so that the final model computes the correct probabilities. To query the model about the probability of a sentence $w_1\, w_2\, \ldots\, w_k$, the following formula can be applied:

$$P(w_1^k) = \prod_{i=1}^{k} P(w_i \mid w_m^{i-1}) \tag{2.15}$$

where $m = \max\{i - n + 1, 1\}$. In the previous bigram example, the probability of the sentence "hello world" would be expressed in the following way:

$$P(\text{``}hello\text{``}, \text{``}world\text{``}) = P(\text{``}hello\text{``} \mid BOS)\, P(\text{``}world\text{``} \mid \text{``}hello\text{``})\, P(EOS \mid \text{``}world\text{``})$$

where BOS and EOS stand for the special characters "beginning of sentence" and "end of sentence". When querying the model, all these probabilities would be known because of the procedure mentioned at the beginning of the paragraph.

RNNs have also shown an improvement in language modeling. Neural networks have shown to work well when dealing with discrete data [5] and were later applied to create language models [6]. Due to their topology, recurrent networks can hold unlimited context capacity, which proves useful in reducing the error rate obtained from the models [34].

Attention-based models have been reported to benefit recognition rate on speech recognition in general [12] or language modeling in particular [32]. The application of the transformer architecture to language modeling in combination with bidirectionality [16] or LSTM cells [56] yields as a result a model which performs better than previous state-of-the-art models.

2.4.4. Decoding

The decoding step consists of finding the most probable sequence of words given the probabilities obtained by the models. It is the part of ASR where both the acoustic and the language models are combined to extract sentences from the acoustic data.

In order to perform decoding, a graph which contains all the HMM states must be built. The graph consists of the unfolding over time[10] of the whole set of HMM states [59]. Edges represent transitions between HMM states, which are not only composed of the usual HMM transitions but can also include transitions from the end state of an HMM which represents the acoustic unit u_a to the beginning state of another HMM which represents the acoustic unit u_b in the following conditions:

- If both acoustic units are found in a word from a certain pronunciation dictionary such that u_a follows u_b. This dictionary is needed in the acoustic model training step and will be properly introduced in Section 4.4.
- If u_a marks the end of a word w_a, u_b marks the beginning of another word w_b, and w_a appears before w_b. This is known as cross-word acoustic unit.

The decoding graph can be graphically expressed through a trellis, represented in Figure 2.8.

The problem of finding the most probable sequence of HMM states given the acoustic sequence can be tackled by using the Viterbi algorithm [55], which computes an approximation to the real solution by computing the most probable state at every frame. To do this, it uses the graph's previous state $t - 1$ when calculating probabilities for the state t.

2.4.5. Evaluation Measures

There are several measures which are usually applied to evaluate the performance of ASR systems.

[10] The exact number of times that the graph must be unfolded corresponds to the number of frames (or acoustic features) taken from the sample. For instance, if a certain acoustic vector sequence consists of 42 acoustic vectors, then the graph will be unfolded a total of 42 times.

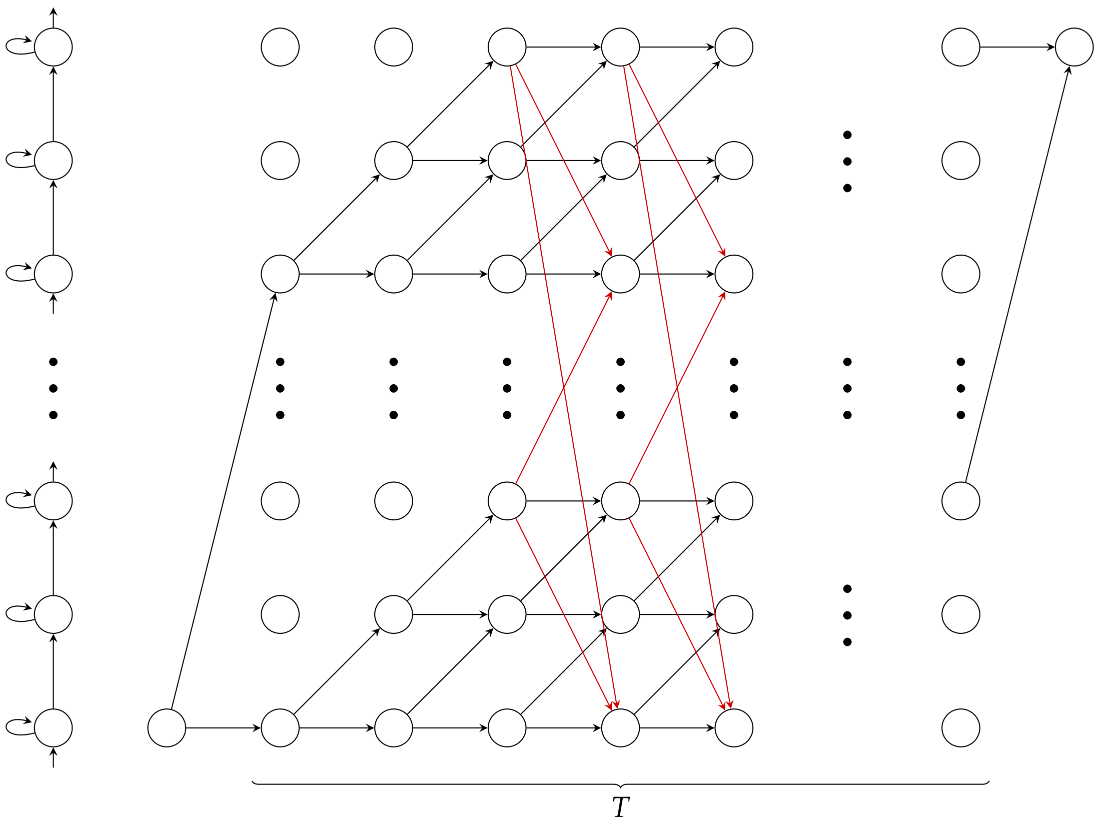


Figure 2.8: Representation of the decoding step by means of a trellis. This trellis is divided into T sections, being T the number of frames or acoustic features from the sample. Black arrows indicate the usual HMM transitions, whereas red arrows indicate transitions which may appear, depending on both the dictionary of words and whether cross-word acoustic units are considered or not.

In this work, the evaluation of the acoustic model when training is represented by means of the frame error rate (FER), which is simply defined as the number of errors that the model obtains when analyzing a given set of frames:

$$FER = \frac{|F_I|}{|F_T|} \tag{2.16}$$

where $|F_I|$ stands for the number of incorrect frames obtained when evaluating, and $|F_T|$ is the total number of frames.

It is common for the language model to be evaluated through the perplexity obtained from the different sets. The perplexity can be seen as the estimated number of words that can follow a given word. Formally, the perplexity PP of a word sequence $W = \{w_1, w_2, \ldots, w_t\}$ is [22, p. 555]:

$$PP(W) = 2^{-\frac{1}{t}\log P(w_1,w_2,\ldots,w_t)} \tag{2.17}$$

where $-\frac{1}{t}\log P(w_1, w_2, \ldots, w_t)$ is an estimation of W's cross-entropy, given a sufficiently long word sequence. Higher perplexity values imply a larger branching factor, which is non-optimal since the model has to look for more words when analyzing a given sentence.

The complete ASR system can be evaluated through the word error rate (WER), which is defined as the minimum edit distance between the original audio transcription and the obtained one. The edit distance generally used is similar to the Levenshtein distance:

it allows insertion, deletion, and substitution at word-level. The WER formula is the following:

$$WER = \frac{I + D + S}{|R|} \tag{2.18}$$

where $|R|$ is the length of the reference transcription, and I, D, and S are the number of insertions, deletions, and substitutions needed for the obtained transcription to become the original one, respectively.

2.4.6. ASR Toolkits

There are few ASR toolkits available. Among the most important are included the Kaldi toolkit[11], the RWTH ASR toolkit[12] or the Julius toolkit[13]. This work uses the transLectures-UPV Toolkit [14] (TLK), a toolkit developed by the MLLP research group[14] at the Polytechnic University of Valencia. This toolkit was developed in the context of the transLectures project (hence its name), aimed at developing software capable of accurately transcribing video lectures.

TLK allows data preprocessing, ASR model training, and decoding. To fulfill this purpose, it contains three high-level tools:

- `tLtask-preprocess` obtains feature vectors from a given audio, following the process described in Section 2.4.1. It can output the data as MFCCs or as filter bank feature vectors, which are generally used to train HMMs and neural networks respectively.
- `tLtask-train` performs standard acoustic model training, receiving as input the corresponding feature vectors. It uses the Baum-Welch and Viterbi algorithms for parameter estimation [14].
- `tLtask-recognise` obtains the most probable utterance corresponding to the attached audio. For this purpose, it does not only need the acoustic model, but also the language model and a pronunciation dictionary.

For a thorough comparison of TLK versus its well-known competitor Kaldi, the reader is pointed to [4].

[11]Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Vesel. The Kaldi speech recognition toolkit. *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*.

[12]D. Rybach, C. Gollan, G. Heigold, B. Hoffmeister, J. Lööf, R. Schlüter, and H. Ney. The RWTH Aachen University Open Source Speech Recognition System. In *Interspeech 2009*, pages 2111-2114, Brighton, U.K., September, 2009.

[13]A. Lee, T. Kawahara and K. Shikano. Julius — an open source real-time large vocabulary recognition engine. In *Proc. European Conference on Speech Communication and Technology (EUROSPEECH)*, pp. 1691–1694, 2001.

[14]The Machine Learning and Language Processing research group specializes in building ASR systems in languages such as English, Catalan, Spanish, French, Slovene, and many others: https://mllp.upv.es/.

CHAPTER 3
Datasets

In this chapter, the two kinds of datasets used to train the models are presented: audio datasets and text datasets. Both types of datasets will be analyzed after explaining some needed concepts which may help understand the choices taken when using these datasets for training the final models.

3.1 Introduction

A dataset is a collection of any kind of data. In this work, data refers to audio files along with their text transcriptions. A corpus, which is a term mainly used in the field of language processing, is a large collection of audio and/or text. Because of the similarity of the definitions, one can take in the context of this work the word "dataset" as a synonym to "corpus".

When training a model, three partitions or sets of the whole data have to be made. Firstly, a training set is needed. This partition will contain data purely used for training the model. Secondly, a development set is created, which aims to optimize training decisions. For instance, it sets the stopping criterion when training by testing the model against this data[1] or helps to model certain parameters before testing (which will be discussed in Chapter 6). Lastly, a testing set is made to obtain a reliable estimation of the model's accuracy.

When a common speaker participates in several audios, speaker dependency may appear between the training and the development partition. It is essential to preserve speaker independence as much as possible since the model could have been trained with words uttered by some speaker who can also be found in the development partition. Then, the system can learn the acoustic features of the voice, seeming to perform better but not being able to generalize. Speaker dependency between the training and the test partition may result in an error rate lower than expected. This topic will be discussed in the relevant corpora.

3.2 Audio Datasets

The audio datasets used in this work are the Clarin-PL Studio Corpus (EMU), the PELCRA family of corpora, the Parlament corpus, the Simple4All Tundra Corpus and the

[1]In essence, if the error rate of this partition tends to rise, it generally means that the model is overfitting: that is, it is learning the training samples and it is not generalizing.

testing results for the PolEval 2019 competition. In this section, each of the mentioned datasets will be described.

- The Clarin-PL Studio Corpus (EMU) [26] is a Polish audio corpus, part of the CLARIN (Common Language Resources and Technology Infrastructure) project, which aims at ”providing easy and sustainable access for scholars in the humanities and social sciences (HSS) to digital language data”[2]. More specifically, it has been provided by the CLARIN-PL, the section of CLARIN devoted to the Polish language.

  It consists of 13 802 short utterances with a total duration of approximately 56 hours. The contents of the corpus are divided in 554 audio sessions recorded by a total of 317 speakers, and every session contains between 20 and 31 audio files. Every utterance has been recorded in a studio, so the resulting audio files are clear from background noise and environmental factors that could have downgraded the final audio quality.

- The PELCRA family of corpora [39] is a dataset of spoken Polish with approximately 262 hours of Polish audios. To use the available data to train the models, an offline version of this dataset was needed, meaning that audio files and transcriptions should be locally downloaded. This could be made thanks to the PELCRA team's effort of compiling resources and separating them into subcorpora[3].

  Unfortunately, it has not been possible to locate the full contents of the corpus, probably because these corpora document audio segments and not the audio as a whole. Therefore, only around 76 hours could be used from this corpus to train the data. Four subsets of the PELCRA family of corpora were used: PELCRA PARL, PELCRA EMO, PELCRA EMI and PELCRA LUZ:

  - The PELCRA PARL dataset is related to the Polish Sejm[4]. This corpus contains full sessions of the previously mentioned body ranging from July of 2016 to February of 2017, along with official transcripts of such sessions in PDF format. Audio durations range from 13 minutes to more than eight hours.
  - The PELCRA EMO dataset consists of interviews focused on people and their emotions, being the interviewer common to all audios. The corpus features 22 audios of a little more than an hour each, every audio being focused on one of the 22 participants. The contents of this dataset were placed on the training partition since the duration was fitting, and the fact that the interviewer participated in every session could have created speaker dependency between partitions.
  - The PELCRA EMI dataset includes audios of Polish emigrants to Scotland. The total audio length of this dataset is a little more than nine hours, so it was appropriate to be used as the development partition.
  - The PELCRA LUZ dataset contains common, semi-scripted interviews on a different range of topics. This dataset is similar to the PELCRA EMO dataset in the fact that there is a common host to all audios and the total duration reaches approximately 20 hours.

[2]Erhard Hinrichs and Steven Krauwer. The CLARIN research infrastructure: Resources and tools for eHumanities scholars. In Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14), pages 1525–1531, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA).

[3]Offline data from the PELCRA corpus can be found at http://pelcra.pl/new/tools_and_resources, in the section Offline Corpora.

[4]The Polish Sejm is in charge of exercising legislative power along with the Senate: http://opis.sejm.gov.pl/en/ (English version).

- The Parlament corpus is originated from an agreement between the Polish Senate and the authors of *System for Automatic Transcription of Sessions of the Polish Senate* [31]. According to Marasek, Koržinek, and Brocki, "the law in Poland mandates that all parliamentary meetings need to be recorded, transcribed, and publicized", which is a great opportunity to create a corpus out of. The total audio duration of this dataset is 97 hours long. Moreover, although utterances are not very long since their mean length is 54 seconds, they are longer than utterances from the Clarin-PL Studio Corpus (EMU).

  The corpus contains a total of 516 parliamentarians and it is already divided into two partitions: a 96-hour-long training partition composed of 506 speakers, and a one-hour-long test partition with 10 speakers. However, the test partition already given was composed of speakers found in the training partition. As a consequence of preserving speaker independence between the system's partitions, everything was placed in the training partition.

- The Simple4All Tundra corpus [50] (Tundra) is a dataset providing nearly 60 hours of audiobooks available at the LibriVox collection[5]. The Tundra corpus currently covers 14 different languages, where audios of every language consist of a single speaker reading a specific book. Since the Polish version is less than three hours long, it was the dataset that covered most of the test set.

- PolEval is an annual competition consisting of creating "natural language processing tools for Polish"[6]. Participants' systems compete in several tasks such as Automatic Bullying Detection (2019), Sentiment Analysis (2017), or Automatic Speech Recognition (2019). The latter task included audio that was less than an hour long. This dataset could be included in the test set thanks to the coordinator of the task, Danijel Koržinek, who released the utterances used in the competition as well as their corresponding transcriptions. Chapter 6 includes a comparison with respect to the participants of this task.

Table 3.1 shows a comparison of every dataset according to their length and the division of the total audio length in the training, development, and testing sets respectively. As can be seen, the total audio duration exceeds the quantity of 200 hours, which is a reasonable quantity of audio to train an ASR system [35].

3.3 Text Datasets

Text datasets used in this work are the following: the National Corpus of Polish, a set of articles from the Polish Wikipedia, the Aranea Polonicum corpus, the Common Crawl dataset, a collection of subtitles from the Opensubtitles website, the ParaCrawl corpus, the Translation Memory of the European Commission's Directorate-General for Translation (*DGT-Translation Memory*), the Europarl parallel corpus and the Polish Parliament corpus.

Audio transcriptions present in the audio corpora were also used as text datasets, with the same usage as that given to their respective audios in the previous section: the transcriptions present in the audio training, development, and testing sets became part of the text training, development, and testing sets, respectively. However, since audio

[5]LibriVox's objective is to release audiobooks with audio recorded from volunteers into the public domain: `https://librivox.org/`.

[6]`http://poleval.pl/`.

Table 3.1: Total audio length divided by individual audio dataset used for the creation of the acoustic model. Datasets are grouped by partition.

Partition	Dataset name	Audio length
Training	Parlament	97 h 33 m
	Clarin-EMU	55 h 49 m
	PELCRA EMO	26 h 53 m
	PELCRA LUZ	20 h 14 m
	PELCRA PARL	12 h 22 m
	Total	212 h 51 m
Dev.	PELCRA EMI	9 h 36 m
Testing	Tundra	2 h 44 m
	PolEval 2019	48 m
	Total	3 h 32 m
–	Total	225 h 59 m

corpora have already been described in the previous section, their transcriptions will not be described in this section.

- The National Corpus of Polish[7] (NKJP) is one of the most extensive text datasets of the Polish language. This corpus contains over fifteen hundred million words extracted from several available media such as books, newspapers, classic literature, or internet texts, which can be queried on their official website. Unfortunately, only a million-word subset of this corpus is available to the public for downloading.

- Wikipedia is one of the largest and most famous sources of information over the Internet. The interest for this free resource lies in the fact that a Creative Commons Attribution-Share-Alike (CC-BY-SA) license is applied to its text. A total of 895 416 articles from the Polish Wikipedia were used, with topics ranging from the AWK programming language[8] to a description of adrenaline[9].

- The Aranea corpora family [7, 8, 44] is a set of corpora which features a total of 22 languages, with special focus on Slovak-Centric languages[10]. The Polonicum corpus is the corpus from the Aranea family that is composed of Polish sentences. This corpus is the second most extensive dataset used, with approximately forty-eight million sentences.

  The Aranea family is accessible to be queried online, much like the NKJP corpus. However, since this work belongs to the academic field, it has been possible for the data manager, Vladimír Benko, to release the data of the Polonicum corpus. From now on, mentioning the Aranea corpus will refer to the Aranea Polonicum corpus.

- The Common Crawl dataset is a text dataset provided by the Common Crawl organization[11]. The Polish part of this dataset was used, which can be found at [58]. This dataset is the most extensive one out of all the datasets considered in this section, with over nine-hundred million sentences. However, due to its crawled nature,

[7]http://nkjp.pl/index.php?page=0&lang=1 (English version).

[8]https://pl.wikipedia.org/wiki/AWK.

[9]https://pl.wikipedia.org/wiki/Adrenalina.

[10]"Languages spoken and/or taught in Slovakia and its neighbouring countries": http://ucts.uniba.sk/aranea_about/.

[11]The Common Crawl organization is a "non-profit organization dedicated to providing a copy of the internet to internet researchers, companies and individuals at no cost for the purpose of research and analysis": https://commoncrawl.org.

some of its contents are not clean. Figure 3.1 shows examples of sentences from this dataset.

Co, do diabła, spodziewaliście się, że zrobię?

0000000000 *nie mozesz lrzywac takich brzydkich słów bo będziesz mlsiałodwłcic sie za to .*

00000linkstart1000000linkend10*klawiatlra wizlalna dla .*

„ „Предмет за хората с увреждания: продължаването на митническите облекчения е съобразно при спазване условията на член 77, параграф 2, алинея втора на Регламент (ЕИО) № 918/83“

:) :) :) :) :) :) :) :) :)

Figure 3.1: Examples of sentences provided by the Common Crawl corpus. Many of the sentences listed in the corpus are coherent (first sentence). However, some sentences have unwanted sequences, most commonly at the beginning (second and third sentences). Finally, there is a small set of sentences that either are in a totally different language from Polish, such as Bulgarian (fourth sentence), or do not make any semantic sense (fifth sentence).

- Opensubtitles[12] is an initiative aimed at collecting subtitles from movies. There are currently 315 779 films whose subtitles are in Polish, making it the third most subtitled language in the website[13]. The text contents of this corpus amount to approximately 45 million sentences, with a more informal aspect than the rest of the described corpora due to their cinematic context.

- The ParaCrawl corpus[14] is a public-domain dataset consisting of text crawled from websites. The corpus is separated into 24 different languages. The Polish language has approximately 8.5 million sentences.

- The DGT-Translation Memory[15] is a dataset property of the European Commission. It is available in 24 different languages, being the Polish language one of them. There are more than 6 million translation units as reported by the European Commission.

- The Polish Parliamentary Corpus [37] (PPC) is a set of public-domain documents extracted from the proceedings of the Polish Parliament, Sejm, and Senate. The data has been extracted from the sessions of the two bodies previously mentioned from 1918 up to the present. The structure of this corpus is similar to the structure of the *NKJP* corpus.

  According to Ogrodniczuk, the PPC is ”among the largest parliamentary corpora worldwide, amounting to approx. 300M words” [37]. However, approximately 478 million words were found when examining the data, which was more than initially reported. This difference may lie in the fact that the corpus is being constantly updated with the latest transcripts of the sessions.

- The Europarl corpus [25] is a parallel[16] dataset which contains sentences from 21 European languages. These sentences have been extracted from the proceedings of the European Parliament. The original release, which included 11 languages, did

[12]`https://opensubtitles.org`.
[13]Last checked: May 11th, 2020.
[14]`https://paracrawl.eu/`.
[15]`https://ec.europa.eu/jrc/en/language-technologies/dgt-translation-memory`.
[16]A parallel corpus contains sentences in a language and its respective translation in other languages.

not include the Polish language. However, the most recent version features a Polish to English parallel corpus.

While the official data of the corpus states that the Polish to English corpus contains around 632 thousand sentences, there were in fact 233 thousand available sentences, as shown in Table 3.2. One reason could be that the used version, obtained from the OPUS parallel corpus[17], compacted sentences in a more coherent way than the original corpus[18].

Table 3.2: Number of sentences and words of every individual text dataset (including audio transcriptions) and total numbers used in the training of the language model, sorted by descending number of sentences. Datasets are grouped by partition.

Partition	Dataset name	No. sents.	No. running words
Training	Common Crawl	920.5 M	14.2 G
	Aranea	49 M	2.3 G
	Opensubtitles	44.7 M	274.3 M
	PPC	11.8 M	478.5 M
	ParaCrawl	8.5 M	149.7 M
	DGT-TM	6.2 M	94.3 M
	Wikipedia	4.9 M	141.6 M
	Europarl	233.6 k	13 M
	Audio transcriptions (training)	87 k	1.1 M
	NKJP	39.5 k	1 M
	Total	1 G	17.7 G
Dev.	Audio transcriptions (dev.)	11 k	96 k
Testing	Audio transcriptions (testing)	1.9 k	29.4 k
Total	–	1 G	17.7 G

[17]The OPUS parallel corpus is a collection of texts from different languages based on open-source data: http://opus.nlpl.eu.

[18]For instance, two sentences in the original Europarl corpus, "9." and "*Zarządzanie europejskimi programami radionawigacyjnymi*" ("Management of the European satellite radio-navigation programs"), became one sentence in the OPUS version of the Europarl corpus: "9. *Zarządzanie europejskimi programami radionawigacyjnymi*".

CHAPTER 4

Acoustic Model Training

This chapter details the tools used when training the final acoustic models, as well as the calls made in order to preprocess the audio samples and the sequence of steps taken to train these models.

4.1 Introduction

The goal of the acoustic model training is to classify the MFCCs or filter bank feature vectors obtained from the data preprocessing step into senones. More specifically, the output of the model when given a frame as an input (or set of frames: see Section 4.6) should be a triphone state (senone), that is, a phoneme that carries the previous and the next phoneme units as context. Senones will be formally defined in Section 4.4.

Three acoustic models are trained in this work: a GMM-HMM model, a DNN-HMM hybrid model, and a BLSTM-HMM hybrid model. As a reminder, these three models are different from each other: it is assumed that the HMM's emission probabilities of the first model resemble a Gaussian mixture, while such an assumption is not made in the second model (these probabilities are modeled by a DNN). The BLSTM implements the memory architecture discussed in Section 2.4.2, and because it is bidirectional, it can model past as well as future context. Both the DNN and the BLSTM estimate the emission probabilities of the HMM models.

The goal of training both the DNN-HMM and the BLSTM-HMM hybrid models is to further perfect the GMM-HMM model's output. For both of these, only the actual networks will be trained, since the HMM part will be reused from the GMM-HMM model. The training of the three models will be detailed in the subsequent sections, as well as the data preprocessing carried on both the utterances.

Prior to training the networks, two alignment steps are needed to be taken, one for each network training. The function of this alignment will be explained in the sections concerning the DNN and the BLSTM training.

Figure 4.1 shows a simplified sequence of actions done to train the acoustic models. First, data preprocessing is applied to the acoustic samples. Then, the GMM-HMM model is trained, and data is aligned by using this model. After this, the DNN is trained with the obtained GMM-HMM alignments, and data is again aligned with the DNN-HMM hybrid model. Finally, the BLSTM is trained with the data coming from the DNN-HMM alignment step.

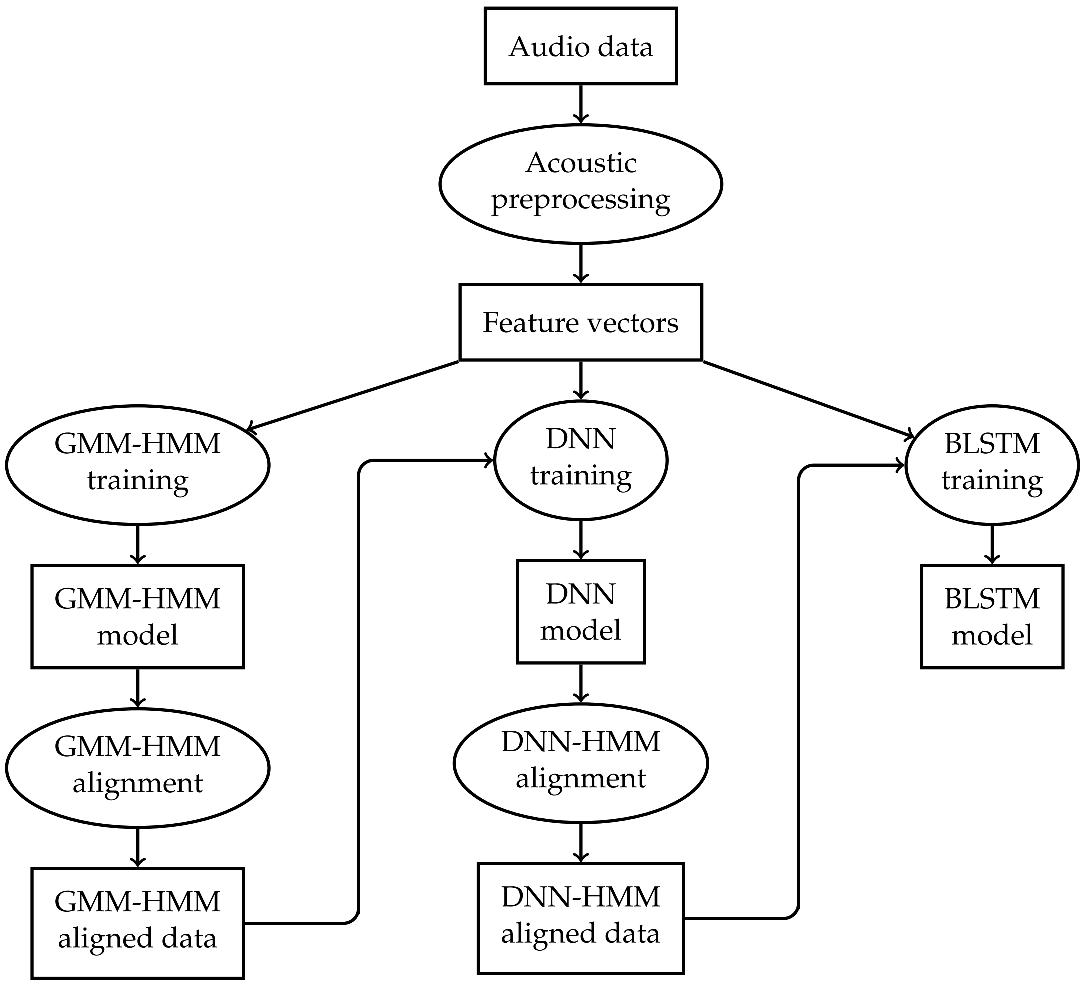


Figure 4.1: Basic sequence of actions performed in order to train the final acoustic models. Circles design actions, while squares reference available data or information at each step. This figure is best followed from left to right, and from top to bottom.

4.2 Training Tools

Three main tools are used to train the acoustic models: TLK, TensorFlow, and Sequitur G2P.

- TLK has already been described in Section 2.4.6. It is used in this work to preprocess audios, train both the GMM-HMM and the DNN-HMM models, and decoding audio files.
- TensorFlow [1] is a machine learning platform available for free which allows creating machine learning models. The created models can adapt to the platform to which the model is intended to be used, ranging from low power consumption platforms to large-scale infrastructures. TensorFlow's source code is available to the public and hosted at Github [1].

  TensorFlow is used in this work to train a BLSTM for substituting the DNN in the DNN-HMM hybrid model, thus theoretically improving the resulting model.
- The function of a grapheme-to-phoneme converter (G2P) is to obtain a word's phonetic transcription given its graphical representation as a word. This is needed in

[1] https://github.com/tensorflow/tensorflow.

ASR to relate the acoustic model (which works at the phoneme level) with the language model (which works at the word level). Figure 4.2 shows an example of the G2P output.

polskiego	p o l s k j e g o	[pɔlskiɛgɔ]

Figure 4.2: Polish word *polskiego*[2](left), its obtained phonetic transcription as the output of the G2P (center) and its IPA phonetic transcription (right). The obtained phonetic transcription does not follow any standards other than the pronunciation dictionary's transcriptions, which are handcrafted by the user.

Sequitur G2P [9] is a data-driven tool developed by the RWTH Aachen University. It requires a previously prepared phonetic dictionary of word pronunciations in order to work, and it works for any language because the only requirement is the pronunciation dictionary. The most basic execution will generate a unigram model, which only looks at a single letter to decide the most fitting phoneme. This basic model can be improved in order to train higher-order models, which may look at the letter's context to decide the most fitting phoneme.

As introduced in Section 2.4.4, a division of a word into phonetic units is needed in order to know which HMM states should be concatenated in the decoding step. However, some words may not be found in the pronunciation dictionary provided when training. Sequitur G2P will be used to craft a G2P model, which in turn will be used to obtain the lexical transcription of any word that does not appear in the original lexicon.

4.3 Acoustic Data Preprocessing

Both the GMM-HMM and the DNN-HMM models use MFCC feature vectors as their input. The BLSTM-HMM model created in this work, however, uses the output of the fourth step previously mentioned (filter bank analysis). Moreover, the filter bank analysis carried out in this type of vectors (filter bank vectors) is more thorough: in this work, a total of 85 filters are used when computing these filter bank vectors.

The acoustic preprocessing described in Section 2.4.1 can be carried out by calling the `tLextract` tool, provided by TLK and internally used by the `tLtask-preprocess` higher-level tool. `tLextract` is used to obtain both MFCC and filter bank feature vectors. The differences between both types of feature vectors are the following:

- MFCC feature vectors have a total of 48 dimensions. Out of these, 15 correspond to the 15 cepstral coefficients extracted from the window frame, while another one is computed as the power of the frame. From these 16 dimensions, the first and second derivatives are computed, which can be seen as the frame's velocity and acceleration respectively. MFCC feature vectors are employed to train both the GMM-HMM model and the DNN.

- Filter bank feature vectors utilize 85 filters for the filter bank analysis. However, no derivative is computed in this case, yielding a total of 85 dimensions per feature vector. This type of vectors are used to train the BLSTM.

The `tLextract` tool expects audio in a 16 kHz mono track. This is achieved by using the `sox` utility, which is able to change both of these parameters by using the `channels`

[2]"Polish".

and `rate` options. Although audios in `wav` format are used in this work, TLK supports a large number of audio formats [14].

Audio transcriptions also have to go through a cleaning process. However, due to the similarity of this process with cleaning corpora used for language modeling, it will be explained in Section 5.3, which details this cleaning for both the audio transcriptions as well as the language model corpora.

4.4 GMM-HMM Model Training

The goal of training the GMM-HMM model is to have an initial set of HMMs, each modeling one triphoneme. This initial model will be used to align the samples, a process that will be explained in the following sections.

Before starting to train the GMM-HMM model, a classification and regression tree (CART) must be created to classify triphonemes. A CART is a type of decision tree in which data is classified by asking closed ("yes/no") questions. This tree can then be easily explored in order to get the queried triphoneme's class. Examples of suitable questions would be "is the phoneme open?" or "is the previous phoneme nasal?".

The CART is automatically created by TLK based on a handcrafted set of rules. All the information needed in order to classify triphonemes is summarized in Tables 4.1 and 4.2, and has been obtained from Wikipedia[3]. The task of mapping each phoneme to its corresponding symbol from the given pronunciation dictionary[4] was done manually.

Table 4.1: Left of each cell: Polish vowels represented as non-IPA phonetic sounds, established by the original lexicon. Right of each cell: IPA phonetic transcriptions of the respective G2P representations. The phonetic sounds have been classified in the different phonetic categories.

	Front	Central	Back
Close	i [i]	I [ɨ]	u [u]
Mid	e [ɛ]		o [ɔ]
Open		a [a]	

Table 4.2: Left of each cell: Polish consonants represented as non-IPA phonetic sounds, established by the original lexicon. Right of each cell: IPA phonetic transcriptions of the respective G2P representations. The phonetic sounds have been classified in the different phonetic categories.

	Labial	Dental/Alveolar	Retroflex	Palatal	Velar plain
Nasal	m [m]	n [n]		ni [ɲ]	
Plosive voiceless	p [p]	t [t]			k [k]
Plosive voiced	b [b]	d [d]			g [g]
Affricate voiceless		ts [ts]	tS [tʂ]	tsi [tɕ]	
Affricate voiced		dz [dz]	dZ [dʐ]	dzi [dʑ]	
Fricative voiceless	f [f]	s [s]	S [ʂ]	si [ɕ]	x [x]
Fricative voiced	v [v]	z [z]	Z [ʐ]	zi [ʑ]	
Tap/Trill		r [r]			
Approximant		l [l]		j [j]	w [w]

This expert information is gathered and given as an input to TLK when training the model to create tied phoneme states, more commonly known as senones. Senones are

[3]https://en.wikipedia.org/wiki/Polish_phonology

[4]The original pronunciation dictionary, required to train the G2P model, was obtained from https://github.com/danijel3/ClarinStudioKaldi.

basic subphonetic units that share acoustic similarity. They are created in order to allow parameter sharing between HMM phoneme states since it is very difficult to estimate these with a limited amount of data [23]. Senones are automatically determined by TLK when generating the CART and can be thought of as the leaves of the CART previously created. Figure 4.3 shows a graphical representation of senone sharing.

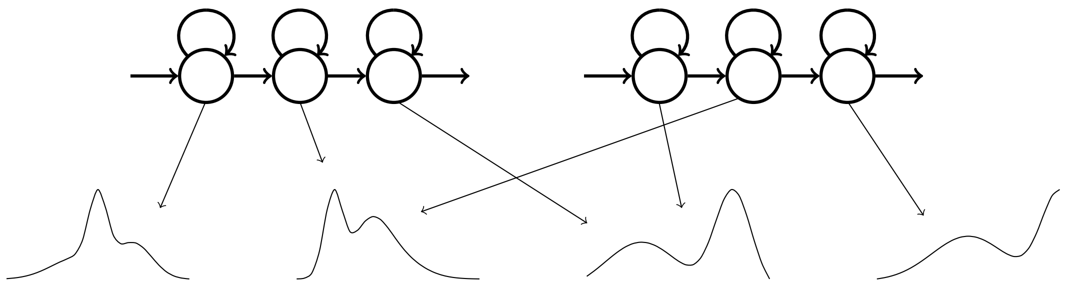


Figure 4.3: Graphical representation of senone sharing by the HMM states. The emission probabilities of each HMM state are modeled by the senones, and each senone may model one or more HMM states' emission probabilities.

To start the training, TLK needs four essential files: a list of the MFCC objects (audio feature vectors), a list of the transcriptions already cleaned for the respective MFCC objects (audio transcriptions), a list of the training words along with the output of the G2P (lexicon), and the expert knowledge required in order to craft the CART phoneme queries.

For the training of the GMM-HMM, the `tLtask-trainghmm` high-level TLK script is used, which internally runs the expectation-maximization (EM) algorithm. This algorithm is an iterative algorithm that tries to optimize the parameters when the statistical model has hidden variables[5]. For more details on this algorithm, the reader is pointed to [59, pp. 33-39].

The output of the training is a set of six tied phoneme models, corresponding to the models trained with a mixture of $2^i : 0 \leq i \leq 5$ Gaussians. TLK also creates a single phoneme model and a triphoneme model, both estimated with a single Gaussian; these are the models needed by TLK before being able to create the tied phoneme models.

The number of senones obtained after the CART creation is equal to 11589. This number will be needed when training both the DNN-HMM and the BLSTM-HMM hybrid models. Furthermore, TLK generates the set of triphonemes found in the training set, grouped by senones: this file will be called `tlist` when referenced later on.

Training these models took approximately five days of CPU resources, which was the only resource used due to the simplicity of the EM calculations. The least time-consuming tied phoneme model trained was the mixture of one Gaussian (four hours), while the most time-consuming tied phoneme model trained was the mixture of 32 Gaussians (four days).

[5]Variables whose information is absent in the training data. For instance, in the case concerning this work, the means μ_i and variances σ_i of each Gaussian component i are hidden variables, since their value is not known when estimating them.

4.5 GMM-HMM Data Alignment

As explained when introducing this chapter, the DNN training needs a frame-to-senone alignment as one of its inputs. This fits in the basic definition of a classification model by taking the frames as the objects and the senones as the classes. This is key in the development of the DNN, because then a proper input (the frame) and its respective desired output (the senone) will be available to train the network. As a consequence, this becomes a problem of finding the most probable class $\hat{s}$ for a given object a, which has already been discussed in Section 2.3.

Five essential files help obtain this alignment: the GMM-HMM model (the most complex one, trained with a mixture of 32 Gaussians), the lexicon, the `tlist` computed in the previous step, the audio feature vectors, and the audio transcriptions. The `tLtask-align` tool, which TLK provides, can be executed for this purpose. The output of the alignment is both the frames in which a certain senone is found within a preprocessed audio and the identifier of the senone. It must be noted that both the training and the development sets must go through this alignment step.

4.6 DNN Training

For the DNN-HMM hybrid model, the HMM's transition probabilities will be reused from the previous step. Only the DNN will be trained, which is the auxiliary structure in charge of modeling the emission probabilities of the HMM. The input to the DNN consists of a given set of 11 consecutive frames (contextual window), and the output will be the probability of recognizing the senone s_i on the set of frames a, or in other terms:

$$P(s_i \mid a) \ : \ 1 \leq i \leq 11589 \tag{4.1}$$

where 11589 is the number of senones obtained in the GMM-HMM training.

Having done the frame-to-senone alignment, the proper training of the DNN can start. The network is composed of an input layer, five hidden layers, and an output layer:

- The input layer has 528 neurons. This layer receives the contextual window, so the number of neurons comes from multiplying the 48 features per frame (for MFCC feature vectors) by the 11 frames of the contextual window.
- All five hidden layers have 2048 neurons each.
- The output layer has 11589 neurons, one for each senone obtained when training the GMM-HMM model. Each neuron i will output $P(s_i \mid a)$, where s_i is the senone i and a is the contextual window.

The input to the DNN training is the alignment mentioned at the beginning of the section (which carries the feature vectors, as described earlier), as well as the network structure just described. The `tLdnn-train` binary from the TLK toolkit can be used to train this neural network. The output of the training is the best model according to the error obtained in the development set.

The DNN was trained on a single NVIDIA RTX 2080 GPU. Its validation error in the training phase converged within a day.

4.7 DNN Data Alignment

As it was the case when training the DNN, the training of the BLSTM needs a previous step of aligning senones to acoustic frames. However, this alignment will be carried out through the DNN trained in the previous step, since it is assumed that it will yield a better alignment. The input to the alignment step is the same as in the previous step with a difference: the model used is now the DNN-HMM hybrid model, instead of the GMM-HMM model. Again, `tLtask-align` can be used to carry out this alignment, yielding the same outputs as in the previous alignment step.

4.8 BLSTM Training

The BLSTM is intended to perform the same function as the DNN in the DNN-HMM hybrid model. However, the fact that the former can store and remember inputs in LSTM cells should theoretically help when recognizing a sample. It must be noted that this network does not need a contextual window since its memory mechanisms allow it to hold acoustic context.

The structure of the BLSTM is similar to the DNN's, with slight differences:

- One of the LSTM layers is in charge of receiving the input vectors.
- There are now four hidden layers (including the previous one) of 1024 LSTM cells each. Because of the bidirectional nature of the network, these four hidden layers extend in both time directions (positive and negative), yielding a total of 512 memory cells for each time direction.
- The output layer has 11589 neurons, which corresponds to the number of senones in the model. This layer's functionality is the same as the DNN's, and its units calculate the same probability as the one described in Equation 4.1.

Training is executed through the TensorFlow API. The input to this training is the alignment performed by the DNN, converted to a set of `TFRecords`[6], as well as the network structure. The output is the set of models obtained when training the BLSTM.

Every model took approximately two hours to be estimated, which made for a total of two days training on an NVIDIA GTX 1080 Ti GPU before the validation error converged.

4.9 Acoustic Model Evaluation

This section intends to evaluate the acoustic models developed in this work. First of all, it is interesting to see the difference in error rate between the DNN and the BLSTM. Figure 4.4 shows the evolution of this error rate with respect to the elapsed epochs.

As can be seen from the previous figure, the DNN converges much faster than the BLSTM. However, it achieves worse error figures, which was already claimed when describing the basis of both models: the main reason resides in the fact that BLSTMs can store memory in LSTM cells and are aware of both past and future, which helps on reaching lower error rates. It could also be the case that the DNN-HMM alignment is more representative than the GMM-HMM alignment.

[6]Binary data used by TensorFlow: https://www.tensorflow.org/tutorials/load_data/tfrecord.

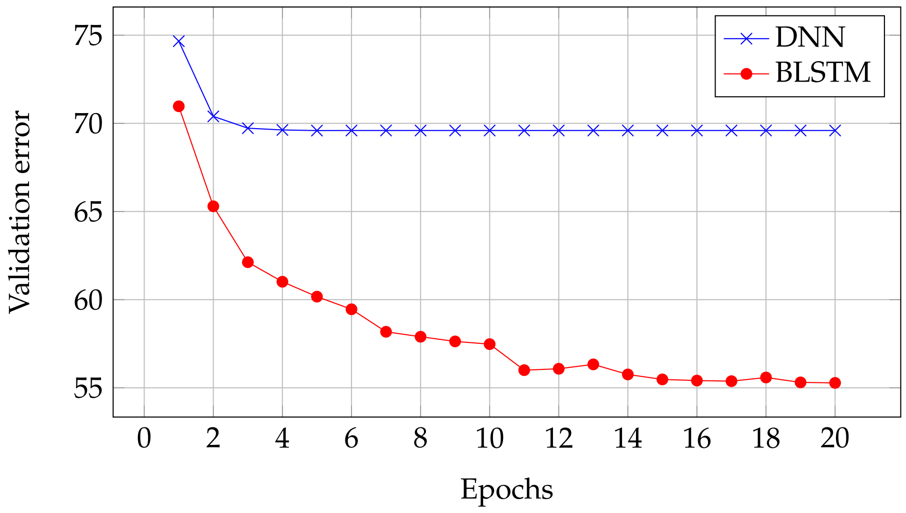


Figure 4.4: Phoneme error rate on the validation set obtained when training the DNN and the BLSTM with respect to each elapsed epoch.

CHAPTER 5
Language Model Training

This chapter describes the procedure carried out to train the final models, as well as the cleaning of the text data used in the creation of the language models.

5.1 Introduction

Three types of language models are trained: an n-gram model (more specifically, a four-gram model), an LSTM, and a transformer. As a reminder, an n-gram model is limited to calculating n-gram frequencies, while an LSTM implements memory thanks to its memory cells, and a transformer implements an attention mechanism that is theoretically supposed to improve the resulting model.

Before training the language models, a cleaning process must be applied to all the text data used in the creation and testing of these models. This process can be seen as data preprocessing, where some general steps to clean the text are followed. However, because of its ad hoc nature, data must also be examined before taking specific cleaning decisions.

Figure 5.1 shows the simplified sequence of actions done to train the language models. First, data preprocessing is applied to the text in order to clean it. Then, the four-gram, the LSTM and the transformer language models are trained. Finally, an optimal interpolation that most benefits the development partition is estimated with respect to the three basic language models.

Theoretically, the three language models could be trained in parallel. However, because it is computationally expensive to train the LSTM and the transformer language models, a subset of the data will be used in order to train the LSTM and the transformer language models. This subset is gathered by following a specific criteria that will be explained in Section 5.5.

5.2 Training Tools

Three tools are used to train language models: SRILM, CUED-RNNLM, and Fairseq:

- SRILM [51] is a language modeling toolkit developed by the SRI Speech Technology and Research Laboratory. It consists of a set of C++ libraries, executable programs, and scripts. This toolkit has been used not only in speech recognition tasks but also in machine translation or even as an introductory tool in teaching.

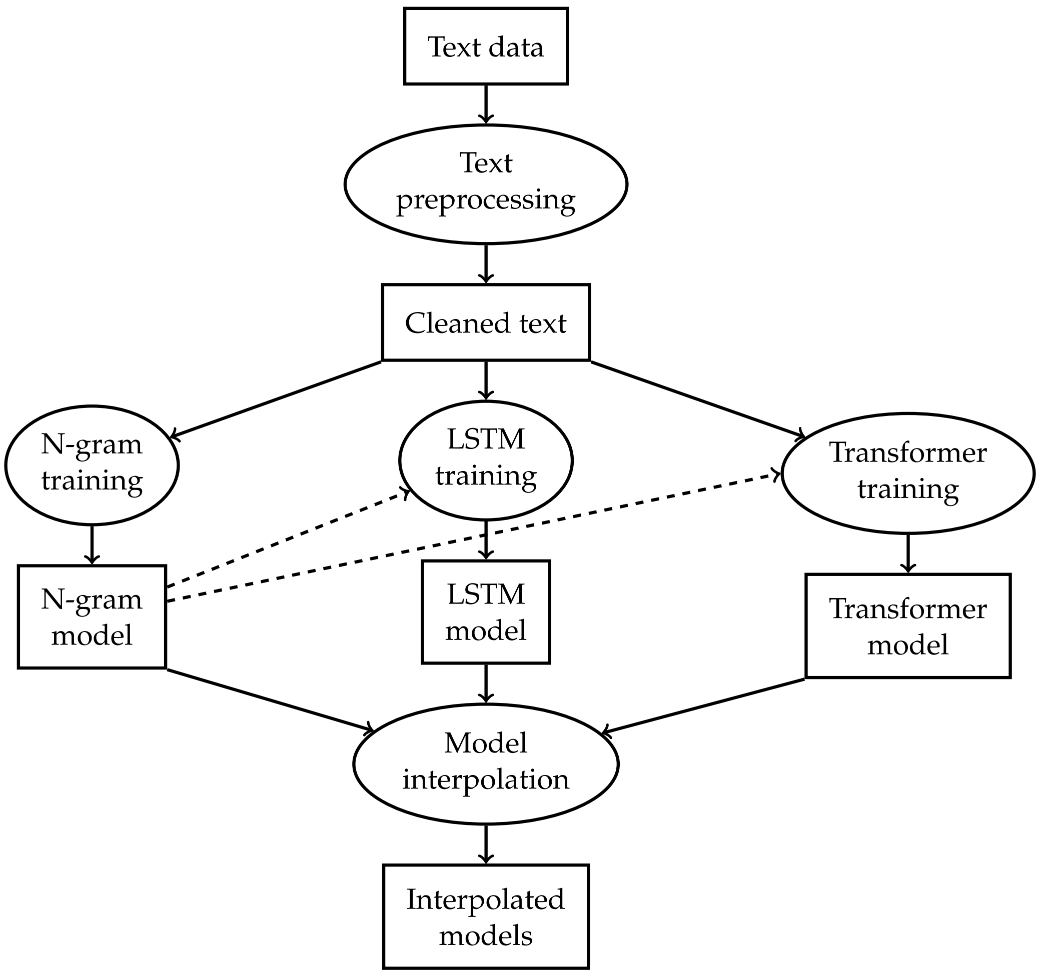


Figure 5.1: Basic sequence of actions performed in order to train the final language models. Circles design actions, while squares reference available data or information at each step.

SRILM creates n-gram language models through previously gathered data. The n-gram order can be specified through a command-line option when executing the programs. This basic functionality can be enhanced through the usage of extensions available on the SRILM website[1]. It also supports the evaluation of the created model by measuring the perplexity by corpus.

The SRILM toolkit contains several binary files and utilities which are used to build the language models. The ones used in this work are described next:

- The `ngram-count` binary calculates the probabilities of the different n-grams of the language model. Essentially, it generates n-gram counts from a given text and creates their associated language model. This binary needs a vocabulary file as one of its inputs, where unique word occurrences found in the text datasets are specified one for each line.
- The `ngram` binary can manipulate the language models obtained with `ngram-count` and obtain several measurements out of them. Examples of actions that can be done are random sentence generation or perplexity calculation with respect to any set of transcriptions given as an input. It can also be used to create the interpolated language model from several independent language models by providing a set of weights. The `ngram` binary also needs a vocabulary file as one of its inputs.

[1] http://www.speech.sri.com/projects/srilm/extensions.html.

  - The `compute-best-mix` script iteratively calculates the best linear combination of several language models with respect to a given text set. These weights are needed in order to create the interpolated language model by the `ngram` binary. The criterion to optimize is the minimization of the perplexity on the text file passed as an input.

  The SRILM toolkit will be used in this work to train the most initial language model: the n-gram language model.

- CUED-RNNLM [11] is a toolkit developed by the Cambridge University that allows training and evaluating an RNN-based language model. In this toolkit, training is performed on a GPU, while evaluation is done on a CPU. Using the GPU to create the language models greatly speeds up the training process, which is done in the toolkit by using the NVIDIA CUDA library for GPU computation. The evaluation of the created language models is done by measuring the perplexity of the test set.

  CUED-RNNLM will be used to train an LSTM as a language model, which theoretically improves performance over the n-gram language model.

- Fairseq [38] is a toolkit that allows custom model training for language modeling, machine translation, summarization, and other related tasks. It is implemented in PyTorch[2] and it allows multi-GPU and multi-machine training.

  Fairseq is an interesting tool to use in this work because it allows the user to train transformer language models. Section 5.6 examines more thoroughly this training.

5.3 Text Data Preprocessing

All text data must go through a preprocessing stage before being used as training data, which includes both the texts used to train the language models and the audio transcriptions. Essentially, no characters other than the letters appearing in the Polish alphabet (non-capitalized) must appear on the text files. This includes numbers, which must be converted into words.

As a reference, there are nine letters which do not appear on the Latin alphabet but are present on the Polish alphabet: ą, ć, ę, ł, ń, ó, ś, ź and ż. Consequently, files cannot be assumed to be in ASCII format. Instead, the cleaning must be done assuming that files are written in a Unicode format.

Three steps are taken to clean both the audio transcriptions and the data from the corpora:

1. First of all, punctuation signs (such as periods, commas, interrogations, hyphens, or quotation dashes) must be removed. This was done in Python through the usage of the `re` module[3].

2. After this step, a conversion from numbers to words is needed. When a number is uttered by a speaker, it is generally desired to see its output as a word rather than as a number. This was done in Python by using the `num2words` library[4], which has support for the Polish language.

[2]PyTorch is a widely known, open-source machine learning framework written in Python: https://pytorch.org/.
[3]https://docs.python.org/3/library/re.html.
[4]https://pypi.org/project/num2words/.

3. The last step is to convert the text to lowercase. This step can be combined with the previous one: those words that are not considered as numbers are converted to lowercase.

The first two steps are not interchangeable because, depending on the way of splitting the sentence, some numbers with punctuation signs around them like 42., (1) or 17-19 could be treated as literal strings instead of numbers, and so the conversion from numbers to text would not be reliable.

It is also a good idea to obtain the vocabulary as well as the alphabet of every corpus after each step to quickly detect unwanted words or symbols and act accordingly.

Cleaning of data which is used for language modeling does not stop at the third step. Due to the crawled nature of some of the corpora, there are sentences written in languages which are not written in the Polish language, being Figure 3.1 a clear example of this. Moreover, there could also be duplicated sentences inside the corpus. Therefore, two more steps are taken to properly clean the text resources:

4. Deduplication is needed so that the model does not receive word-by-word duplicates. It was achieved by using the `sort` utility along with the `-u` flag, which sorts data and removes duplicates.

5. Completely foreign phrases need to be removed because the system is intended to recognize the Polish language only. For this purpose, the Python `langdetect` library[5] was used, which detects the language in which the sentence is written and has support for the Polish language.

The `langdetect` library makes it possible to remove those sentences which are not *completely* in the Polish language. However, there may be *fragments* of sentences in other languages, mixed with sentences mainly in the Polish language. For instance, the sentence 龟尾钴*wrzedsiebiorstwo wrodlkcyjne* is recognized as Polish with an average probability of 99.999% obtained from five different measurements, even though it has characters belonging to the Chinese language.

As can be seen, this situation is more difficult to handle. However, when training the models, only a subset of the vocabulary is chosen (the most frequent unigrams), and those words that lie outside this vocabulary are treated as unknown words or UNK. This does not completely solve the situation, but it is an effective way to deal with it, assuming that foreign words rarely happen. Vocabulary reduction is also useful to reduce the number of model probabilities (for n-gram based language models) or network parameters (for network-based language models).

Finally, as expected, the Common Crawl corpus was too big, so a subset of 100 million sentences was chosen at random through the usage of the `shuf` command.

Text cleaning considerably reduced data in the training partition. Table 5.1 shows the total sentences after this procedure and the relative reduction with respect to the original data (already shown in Table 3.2). It must be noticed that the training, development, and testing transcriptions from the audio datasets have a relative reduction of 0% since neither the removal of foreign sentences nor the deduplication steps were applied to these transcriptions.

[5]https://pypi.org/project/langdetect/.

Table 5.1: Number of sentences after data cleaning and relative reduction of every corpus with respect to the original data, sorted by descending relative reduction.

Dataset name	No. sents.	Rel. reduction (%)
Common Crawl	100 M	89.1
Aranea	24.7 M	49.6
DGT-TM	3.6 M	42.4
Wikipedia	3.6 M	27.2
Polish Parliamentary Corpus	9 M	23.9
ParaCrawl	7.1 M	16.1
Opensubtitles	41.1 M	8.1
Europarl	220.9 k	5.4
NKJP	37.5 k	5.2
Audio transcriptions (training)	87 k	0.0
Audio transcriptions (development)	11 k	0.0
Audio transcriptions (testing)	1.9 k	0.0
Total	189.5 M	81.9

5.4 N-gram Model Training

The final goal is to train a four-gram language model for every available text corpus in order to obtain an interpolated four-gram language model. This interpolation will be calculated by obtaining the weights that most benefit the perplexity of the development partition. Before starting the four-gram language model training, a unigram language model needs to be trained for every dataset, which will then be used to train the four-gram language models.

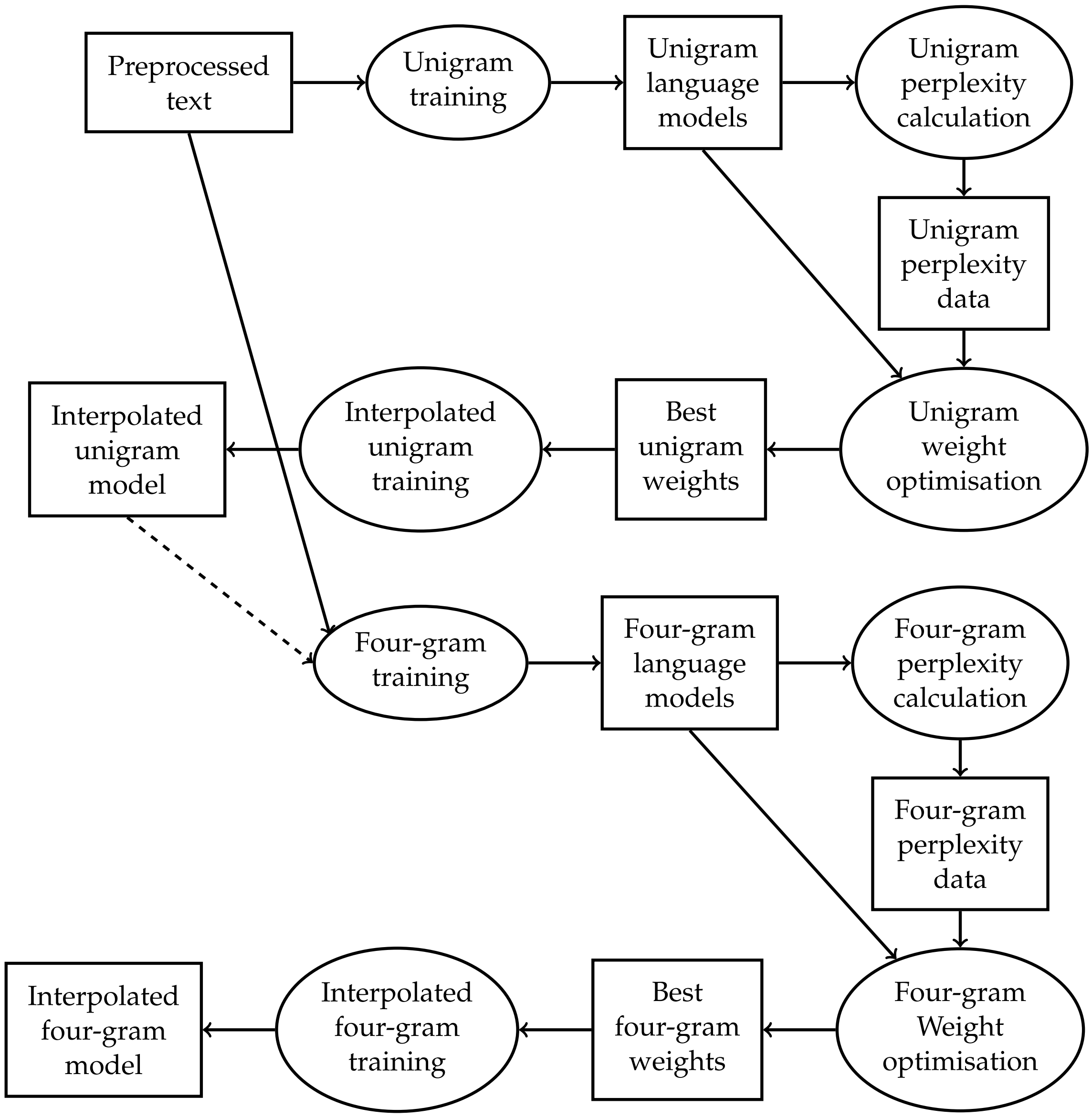
Figure 5.2 depicts the sequence of steps taken to complete this training. In essence, a unigram language model has to be created for each text dataset in the training set. Then, an interpolated unigram language model will be obtained by optimizing the perplexity of the set of unigram language models with respect to the development set. With this data, a four-gram language model will be built for every dataset in the training set, and their perplexities with respect to the development set will be optimized in order to obtain the final interpolated four-gram language model.

The complete steps taken to compute the interpolated four-gram language model are:

1. An n-gram of order one has to be computed for every text dataset used. For this, the `ngram-count` binary is executed, along with the option `-order` and the n-gram order (`-order 1` in this case, since unigrams are being built). As a result, ten n-gram language models of order one are obtained.

2. Individual perplexities have to be calculated for every language model computed in the previous step with respect to the development data. For this, the `ngram` binary needs to be executed a total of ten times, indicating the `-ppl` option along with the development text file. The result of this step is a total of ten separate files with the language model perplexities computed when facing the development set.

3. The best linear combination of the language models with respect to the development data must be calculated. This can be done through the `compute-best-mix` script. It only needs as an input the perplexities computed in the previous step. As a result, a probabilistic vector consisting of the linear combination coefficients is given. Every weight is chosen so that the perplexity is minimized in the devel-



Preprocessed text
Unigram training
Unigram language models
Unigram perplexity calculation
Unigram perplexity data
Interpolated unigram model
Interpolated unigram training
Best unigram weights
Unigram weight optimisation
Four-gram training
Four-gram language models
Four-gram perplexity calculation
Four-gram perplexity data
Interpolated four-gram model
Interpolated four-gram training
Best four-gram weights
Four-gram Weight optimisation

Figure 5.2: Simplified sequence of steps taken in order to train the final n-gram language models. Circles design actions, while squares reference available data or information at each step. This figure is best followed starting from the top left node.

opment set when using the final interpolated language model. Table 5.2 shows the obtained weights for every language model.

4. The interpolated language model has to be obtained. For this, the `ngram` binary is executed again, this time providing the language models computed at the beginning, as well as the weights obtained in the previous step, the vocabulary, and the n-gram order. The result is the interpolated unigram language model.

5. Steps 1 to 4 must be redone but changing a few parameters in order to consider the n-gram order as four. First, the `ngram` binary now has to be executed with the `-order 4` parameter. Then, the vocabulary file must include only the most frequent words so that the percentage of out-of-vocabulary (OOV) words is on the order of 1%: in this case, nearly 250 thousand words. Every other word will be treated as an unknown word (UNK). The most frequent words are obtained from the interpolated unigram language model.

Table 5.2: Weights obtained for each language model when computing the interpolated n-gram language model, rounded to the fourth decimal digit. Both the weights of the order one and four n-gram language models are shown. Weights equal to 0 had a value lower than 1e-3, but were not zero. Data is shown sorted by descending four-gram weight.

Language model	Unigram weight (%)	Four-gram weight (%)
Audio transcriptions (training)	63.8	47.4
Aranea	3.2	21.0
Opensubtitles	32.6	19.0
Common Crawl	0.0	5.6
Polish Parliamentary Corpus	0.2	3.4
ParaCrawl	0.1	3.3
NKJP	0.1	0.3
DGT-TM	0.0	0.0
Europarl	0.0	0.0
Wikipedia	0.0	0.0

The fact that the training transcriptions obtained that much weight can be linked to the fact that part of the audio transcriptions, as well as the development set, also came from the PELCRA family of corpora. This family has a non-standard way of specifying hesitation through words such as "yy", "mm" or "ee", which is frequently repeated throughout all audios from the PELCRA corpora. It is also logical to say that the PELCRA EMI shares a similar structure to other PELCRA subcorpora.

Moreover, the reason why the Aranea and the Opensubtitles corpora had an important weight on the final models could be linked to the used language type: both consist of a more colloquial language, in contrast with the formal language which can be observed in the rest of the corpora. It must be remembered that the development set (in essence, the PELCRA EMI dataset) is composed of friendly interviews with emigrants.

After this last step is finished, the interpolated four-gram language model will have been obtained. This final model also contains all unigrams, bigrams, and trigrams found. The perplexities can be (optionally) calculated with respect to the development and the testing partitions to check the model's performance. This step is needed if an interpolated language model is going to be used (see Section 5.7). Table 5.4 shows the perplexities obtained for each language model with respect to the n-gram order.

In order to keep only the most relevant n-grams, a pruning step is introduced. Pruning deletes the n-grams whose respective probabilities fall below a probability threshold ϵ. Two thresholds are chosen: ϵ = 2e-08 and ϵ = 4e-10, used for the interpolation with the rest of the language models and the recognition (see Chapter 6), respectively.

Completing these steps took approximately two days. The final interpolated model occupied 63 GiB of disk space, most of which was composed of n-grams that rarely appeared. The pruned interpolated n-gram models consumed 132 MiB and 2.5 GiB of disk space for the values of ϵ = 2e-08 and ϵ = 4e-10, respectively.

5.5 LSTM Model Training

The LSTM is trained in order to create a language model which can yield the probability $P(w_i \mid h_i)$ in a presumably more reliable way than n-gram language models since the former implement memory in the way of LSTM cells.

For the LSTM language model training, a subset of a thousand million words is used due to the high computational cost of training the network. This implies a reduction of roughly 75% with respect to the data used to train the n-gram model. To obtain this amount, random samples of the corpora were obtained by taking the proportional part according to the weights of every language model obtained in the four-gram interpolation step, already shown in Table 5.2. For instance, a total of 190 million words were extracted from the Opensubtitles corpus, since its weight on the interpolated four-gram language model accounted for 19% of the total[6].

To train an LSTM language model with CUED-RNN, three key files are needed: the training text, the development text (the same as in the n-gram model training), and the vocabulary, which consists of the 250 thousand words obtained when training the n-gram model. The toolkit needs the vocabulary in a `wlist` format, which consists of the vocabulary words and an index preceding it. This index starts at zero and is incremented by one for each word.

The network architecture is composed of an input layer with about 250 thousand neurons (one for each word in the vocabulary), a projection layer composed of 256 neurons, a hidden layer composed of 2048 LSTM units, and an output layer with the same size as the input layer. The function of each layer is explained in the following. For more details on this network architecture, see [47]:

- The input layer receives the context h_i preceding the word w_i as a one-hot encoded vector. This type of vectors are $|V|$-dimensional: they have a one in the dimension i such that the word found in the context at time t is w_i. The rest of the vector consists of zeros. Note that the LSTM does not receive the full context immediately. Instead, words are inputted one by one, and the network is in charge of building the context.

  It must also be highlighted that w_i is not received: this is because is not needed in the input layer but on the output layer, since the network's function is to calculate $P(w_i \mid h_i)$, and then nothing can be known from w_i in advance.

- The projection layer converts the one-hot encoded vector into a continuous space composed of 256 dimensions. This layer acts as a linear projection layer, projecting the layers by multiplying the one-hot encoded vector by a 250k $\times$ 256 matrix[7] whose weights are learned in the training process.

- The LSTM layer incorporates 2048 LSTM units, whose functionality has already been described in Section 2.4.2.

- The output layer yields an estimation of $P(w_j \mid h_i) \quad \forall\, j \in \{1, 2, \ldots, |V|\}$. However, only the probability $P(w_i \mid h_i)$ is of interest in this step. This estimation can be obtained by indexing the output vector by the word index, present in the `wlist` given to CUED-RNNLM.

The network was trained with additional parameters, such as noise-contrastive estimation (NCE). The idea behind NCE is to have artificially generated noise compared against the real data, and thus learn by comparing both sources of data so that properties of the real data can be learned in the form of a statistical model [19]. This technique works well in combination with language models since they tend to not overfit data while yielding better results than conventional approaches [29].

[6]It was not possible to gather 474 million words from the audio transcriptions due to its reduced size. Instead, the remaining part was extracted from the Aranea corpus, since it obtained the second best weight.

[7]Assuming that the one-hot encoded vector is given as a column vector, and the linear projection is calculated as $p = Mx$, where M is the matrix and x is the one-hot encoded vector.

Training the network with CUED-RNNLM consists of executing the `rnnlm.cued.vX` binary, where `X` corresponds to the version used, with the appropriate options. The language model can also be evaluated with respect to the development set by executing the `rnnlm.cued.vX.eval` binary. In the case of this work, the `1.1` version was used.

The model was trained with an NVIDIA GTX 1080 Ti GPU, and the training process took approximately three and a half days per epoch (that is, a single run through the whole data), with a total of five epochs performed.

5.6 Transformer Model Training

The transformer model also outputs the probability $P(w_i \mid h_i)$. However, it does so by computing attention on a predefined history size $|h_i|$, thus being able to learn long-term dependencies more effectively than RNNs.

The transformer model training is done through the Fairseq toolkit. For this, three files are needed: the training text, the development text, and the vocabulary with the most frequent words. These three files are the same as in the LSTM model training, with the only difference that the vocabulary is not needed as a `wlist`. Instead, the raw vocabulary of one word per line must be used.

Training a transformer model through Fairseq consists of four steps:

1. Before starting to train, the data has to be preprocessed. This can be done by calling the `fairseq-preprocess` command-line tool and providing it with the three files already mentioned. The outcome of this process is the binarized version for the training and the development texts (for later use on the training).
2. The `fairseq-train` command-line tool is called to start the training. As a result, a set of checkpoints are created, which mark the evolution of the model through time.
3. It is recommended to compute the average model of the obtained checkpoints. This can yield an improvement in the resulting model.
4. It is also recommended to calculate the perplexity with respect to the development set to check the transformer model's performance. As in the previous cases, this step is mandatory if an interpolated language model is going to be used later on.
5. Lastly, the model is compiled to achieve more speedup when using it.

Before training the transformer model, some parameters must be specified. Some of the most interesting parameters to mention here are the number of layers and the transformer's dropout. The dropout indicates the probability that a neuron and its adjacent weights are not used when processing a given input. This technique can also be interpreted as adding noise to the network and has proven to prevent overfitting [49]. In this case, the transformer has eight layers and a dropout probability of 10%.

The obtained perplexities can vary according to the history available by the transformer. The transformer's history is the equivalent term of the n-gram's context: it is the maximum amount of words where the transformer will compute attention. This parameter is modifiable after training. Section 5.8 explores the usage of this parameter for the model's evaluation.

The transformer language model was trained with two NVIDIA RTX 2080 GPUs. This change in the training hardware was noticeable since each epoch took approximately a day to finish. The model completed a total of six epochs.

5.7 Interpolated Model Estimation

After having trained the three language models, and once the perplexities of each language model have been obtained with respect to the development set, the last task that remains is to interpolate the language models. This procedure is similar to the process performed when interpolating the n-grams of order one and four, in the third step of the n-gram model training: By choosing the best weights of each basic language models that most benefits the development set in terms of perplexity, it is assumed that this choice will also help in general.

Choosing the optimal weights is done (again) through the `compute-best-mix` script, provided by SRILM. As a reminder, this script only needs as inputs the files where the perplexity of the development set is calculated. A total of four interpolated language models were computed, corresponding to the four combinations of language models.

Table 5.3: Percentage of contribution from each individual language model to the four combinations of interpolated language models.

Language model	Transformer (%)	LSTM (%)	N-gram (%)
LSTM/n-gram	–	63.5	36.5
Transformer/n-gram	67.0	–	33.0
Transformer/LSTM	67.0	33.0	–
Transformer/LSTM/n-gram	53.8	17.5	28.7

5.8 Language Model Evaluation

Table 5.4: Perplexities obtained for each language model after being tested against the development transcriptions, rounded to the second decimal digit.

Language model	Unigram perplexity	Four-gram perplexity
Aranea	2199.6	636.8
Audio transcriptions (training)	1275.6	494.7
Common Crawl	2891.1	627.4
DGT-TM	16694.9	7804.4
Europarl	5031.4	2715.3
NKJP	1953.5	880.1
Opensubtitles	1597.3	667.7
ParaCrawl	3192.5	1064.5
Polish Parliamentary Corpus	3193.3	1055.4
Wikipedia	6501.2	2809.3

As established in the second step of the n-gram language model training, every individual language model created needs to be evaluated against the development transcriptions. Table 5.4 shows the perplexities obtained in this process. It can be seen that incrementing the n-gram order from one to four benefits the resulting language models, as the perplexity decreases. These values are strongly related to the weights obtained when training the n-gram interpolated language model, already shown in Table 5.2.

As for the transformer language model, an exploration of the history parameter is done to study the best value for this parameter. A higher history implies a lower perplexity since the attention mechanism can extract information from a wider context. Figure 5.3 shows the evolution of perplexity with respect to the history chosen.

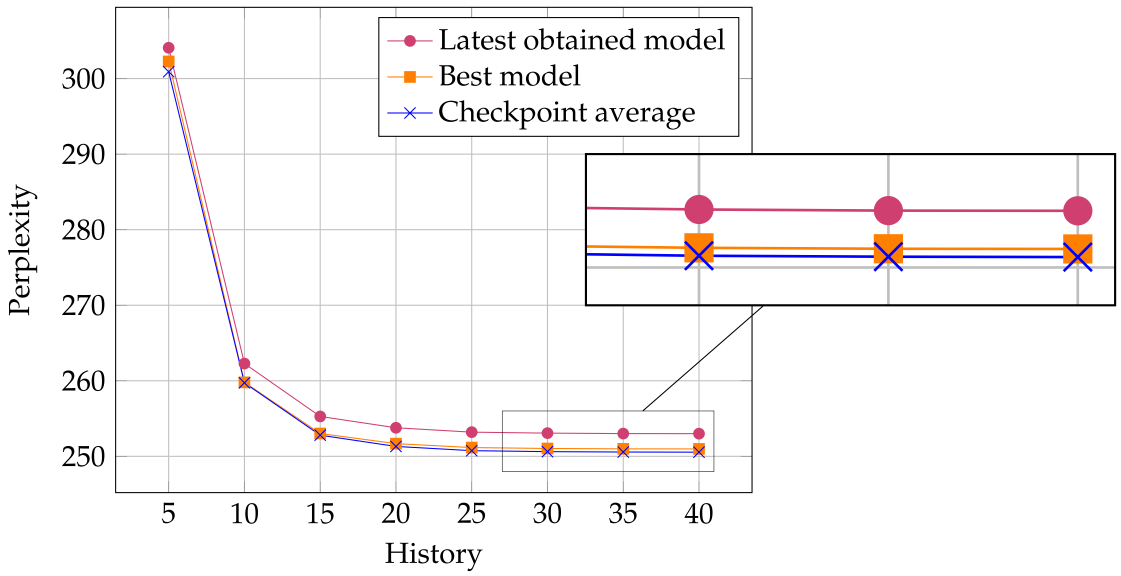


Figure 5.3: Perplexity obtained in the development set with respect to the history used when evaluating the transformer model. The checkpoint average was computed as the average of the last four checkpoints used.

The perplexity obtained falls quickly, and it stabilizes when considering around 20 words in the history. This makes sense considering that 90% of the data from the development set falls at or under a word length of 18 (the mean sentence length is around eight words). As can be seen, more history is beneficial for the model, since it can study more context when focusing on a given word.

Language models are also validated with respect to the development set when training. Figure 5.4 shows a comparison of the three basic language models with respect to the training epochs.

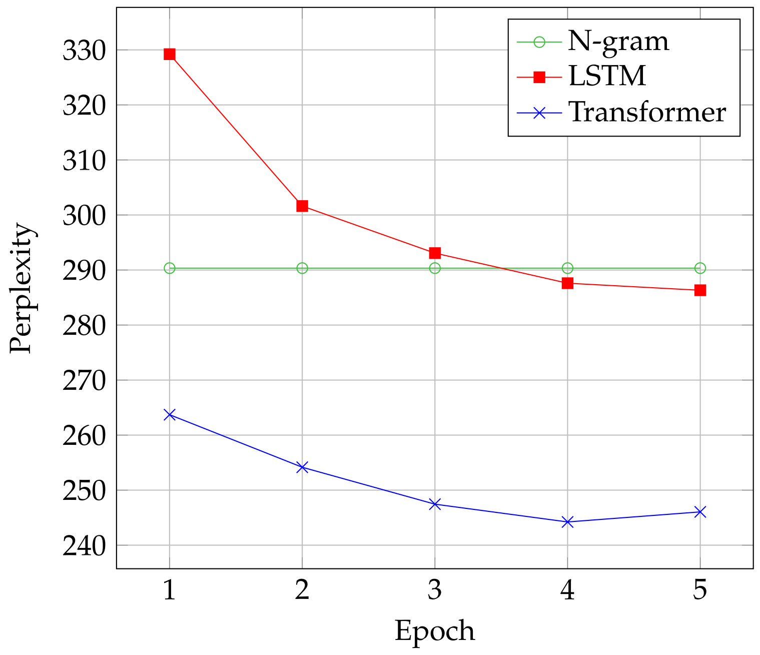


Figure 5.4: Comparison of the n-gram, the LSTM and the transformer language models against the development partition. Data is plotted with respect to the epochs elapsed. The n-gram language model's performance is a flat line because an n-gram language model only goes through the whole data once. Only the first five results from the transformer language model are shown.

Finally, the interpolated language models were obtained. A total of four interpolated model combinations were obtained. Table 5.5 shows the final perplexity obtained when using both the basic and the interpolated language models with respect to the development set. The n-gram perplexity is higher than the one reported in Figure 5.4 since here the perplexity corresponding to the n-gram pruned for calculating interpolation weights is shown, which removes probabilities lower than $\epsilon = 2\text{e-}08$. The weights used for the language model interpolations are the ones already shown in Table 5.3.

Table 5.5: Perplexities obtained when analysing the development set with respect to every language model developed.

Language model	Perplexity
N-gram	366.9
LSTM	293.0
Transformer	250.6
LSTM/n-gram	234.8
Transformer/LSTM	234.6
Transformer/n-gram	202.5
Transformer/LSTM/n-gram	199.0

The best language model obtained was the interpolated transformer/LSTM/n-gram language model, followed by the interpolated transformer/n-gram language model. Interpolated language models obtained better perplexities overall than basic language models.

CHAPTER 6
ASR Evaluation

In this chapter, the created models are evaluated on the development and the test set. Moreover, the fully developed system will be tested against other ASR systems which participated in the PolEval 2019's ASR contest to check its competitive performance. The system will also be tested with respect to the Tundra corpus against Google's speech-to-text system. As a reminder, the development set covered the PELCRA EMI subcorpus (with a conversational nature), and the test set included two datasets: the Tundra corpus (belonging to a literary domain), and the PolEval 2019's ASR contest (belonging to a political domain).

6.1 Introduction

The fully developed ASR system needs to be tested on both the development and the test partitions to evaluate its performance. Before that, a set of experiments must be carried out in order to find the most suitable combination of model parameters. These experiments are performed on the development set.

Two of the most relevant parameters to tune when performing experiments are the grammar scale factor (GSF), which scales the weight of the language model with respect to the acoustic model, and the word insertion penalty (WIP), which helps the system control word insertion and deletion [60]:

$$\hat{w} = \operatorname*{argmax}_{w \in L} \log P(a \mid w) + \alpha \, \log P(w) + \beta \, |w| \tag{6.1}$$

where α refers to the GSF and β refers to the WIP. It is frequent to use these parameters in conjunction with the log-likelihood formula for ASR, which is obtained by applying a logarithm to Equation 2.6. It must be noted that, according to the previous equation, a positive β value implies that the system is rewarded for obtaining a higher number of words, while a negative β value rewards the recognition of a lower amount of words.

Variations in both the GSF and the WIP can yield significant differences when recognizing audio samples, so they are usually explored when evaluating the system.

6.2 Parameter Tuning Experiments

An initial exploration of this parameter took four initial GSF values: 5, 10, 15, and 20. With these results, a finer exploration was performed with respect to the best performing GSF value. Table 6.1 shows both the initial and the finer experiments with this parameter on the development set.

Table 6.1: Word error rate obtained in the development set for each language model with respect to the GSF parameter. The acoustic model used for the evaluation was the BLSTM-HMM hybrid model. Since initially the best performing value was $GSF = 5$, a finer exploration taking the values of $GSF \in \{3, 4, 6, 7\}$ was made.

Language model	GSF							
	3	4	5	6	7	10	15	20
N-gram	**42.3**	**42.3**	42.8	43.8	45.0	50.3	60.5	70.2
LSTM	41.0	**40.8**	41.5	42.3	43.7	49.0	59.7	68.5
Transformer	40.9	**40.5**	40.9	41.7	42.9	48.0	57.8	67.2
Transformer/LSTM/n-gram	41.0	**40.7**	41.1	41.8	42.8	47.6	57.5	67.3

Table 6.2: Word error rate obtained in the development set for each language model with respect to the WIP parameter. The acoustic model used for the evaluation was the BLSTM-HMM hybrid model. All the experiments shown in this table have taken the value of $GSF = 4$.

Language model	WIP						
	-5	-3	-1	0	1	3	5
LSTM	42.1	41.6	41.1	40.8	40.6	40.3	**40.1**
Transformer	41.6	41.1	40.7	40.5	40.4	40.1	**39.9**
Transformer/LSTM/n-gram	41.9	41.4	41.0	40.7	40.5	40.1	**39.9**

As can be seen, the best performing value for the GSF parameter is GSF $= 4$. The transformer language model obtains the least WER (40.5%), closely followed by the interpolation of the three basic language models (40.7%) and the LSTM language model (40.8%).

Then, an exploration of the WIP parameter with respect to the best performing value of the GSF is done to the four models, since they were relatively close in terms of WER. Table 6.2 summarizes these experiments.

The best value of the WIP parameter is $WIP = 5$, which achieves a WER of 39.9% in both the transformer model and the interpolated model, a 40.1% in the LSTM model.

After this experiment, the system is ready to be evaluated with respect to the test set by using the best performing model: the transformer language model with the parameters $GSF = 4$ and $WIP = 5$. Because the experiments report that both the transformer model and the interpolated model perform equally in the development set, the transformer model is (arbitrarily) chosen as the language model that helps recognize the test set.

However, before that, for expositional purposes the system's WER with respect to each one of the interviews from the PELCRA EMI dataset (which covered the whole test set) will be displayed. Table 6.3 gathers this data. In this table, a manual background noise check has been carried out by listening to random interview segments: *none* means that no background noise was identified, *single conversation* means that a single speaker or group of speakers were talking in the background in an ordered manner (such as a conversation, monologue, or sound coming from a television), and *noisy environment* means that there were many sources of background noise (such as in a cafeteria, playground, or street).

Audios where there was no background noise tended to have a lower error rate than those where some kind of noise is identified. There are a few outliers such as interviews 9 and 22. This high WER reported could correspond to the fact that in interview 9 background music was loud and could be clearly understood, while in interview 22 the voice of the interviewed person may have blended in with the background noise.

Table 6.3: Development set's WER separated by each of the 22 interviews from the PELCRA EMI dataset.

Interview no.	Background noise	WER (%)
1	None	23.7
2	None	30.0
3	Noisy environment	23.7
4	None	21.7
5	None	28.4
6	None	38.7
7	Noisy environment	38.6
8	Noisy environment	47.5
9	Noisy environment	73.8
10	None	35.9
11	Single conversation	48.6
12	Single conversation	44.0
13	Single conversation	26.9
14	Single conversation	50.1
15	Noisy environment	36.5
16	Single conversation	32.1
17	Single conversation	36.4
18	None	24.1
19	None	36.9
20	Noisy environment	40.9
21	Single conversation	55.1
22	Noisy environment	84.0

Table 6.4: WER obtained with respect to each of the test set datasets. The minimum amount of insertions, deletions, and substitutions obtained when calculating the WER is also shown.

Dataset name	Insertions	Deletions	Substitutions	Ref. length	WER (%)
PolEval 2019	199	227	482	5728	15.9
Tundra	3973	1225	5777	23690	44.8

Moreover, due to the conversational nature of the dataset, in some of the audio segments the main speaker is often interrupted. However, these interruptions are not captured in the text transcriptions. Instead, transcriptions only have the sentences uttered by the main speaker.

6.3 Final System Evaluation

For clarity, the test set will be separated in the two datasets which compose it: PolEval 2019 and Tundra. Table 6.4 shows the WER obtained, as well as the insertions, deletions, and substitutions that the system made for each dataset.

To conclude this chapter, the complete ASR system developed in this work has been compared against other diverse systems. The results are shown in Tables 6.5 and 6.6.

The WER from Google S2T was obtained by using the Google Cloud Platform's Speech-to-Text API[1], and then extracting the most probable transcription.

[1]Google Cloud Platform: https://cloud.google.com/. Cloud Speech-to-Text API: https://cloud.google.com/speech-to-text.

Table 6.5: Comparison of the ASR system developed in this work against other systems which participated in the PolEval 2019 contest. All data was obtained from the PolEval website, except for the systems named "our system", corresponding to the system developed in this work, and "Google S2T", which corresponds to Google's speech-to-text system. All other results have been obtained from http://2019.poleval.pl/index.php/results/. Last accessed: July 1st, 2020.

System name	WER (%)
GOLEM	12.8
Our system	15.9
Google S2T	19.0
ARM-1	26.4
SGMM2	41.3
tri2a	41.8

Table 6.6: Comparison of the ASR system developed in this work against Google's speech-to-text system with respect to the data available from the Tundra corpus.

System name	WER (%)
Google S2T	30.6
Our system	44.8

Table 6.7: Overall comparison of the ASR system developed in this work against Google's speech-to-text system with respect to the test set.

System name	WER (%)
Google S2T	30.3
Our system	40.4

The system developed in this work outperforms Google S2T by 3.1% on the PolEval 2019 contest. Moreover, it would have reached second place in the PolEval 2019 contest. However, it falls behind Google S2T by 14.2% when tested on the Tundra corpus, and by 10.1% when the whole test set is considered. This WER difference could be caused by the language model: the transformer language model reports an average perplexity of 954 in the Tundra corpus, while the same measurement is equal to 93 in the PolEval 2019 dataset. This may in turn be caused by the change in domain: words such as "jesień"[2] or "gospodarstwie"[3], which were found in the Tundra dataset, belong to a literary domain and are not used much in everyday language, let alone politics.

[2]"Fall" (as in the season).
[3]"Farm".

CHAPTER 7

Conclusions

This work has explored the process of creating a Polish ASR model from scratch. The tools used in the work as well as both the audio and text datasets have been described. Both acoustic models (GMM-HMM, DNN-HMM, and BLSTM-HMM) and language models (n-gram of order four, LSTM, and transformer) have been developed. The result of this work is a fully functional Polish ASR system, which has been tested with respect to speeches belonging to the literary and political domain. Moreover, it has been compared against other systems that participated in the PolEval 2019 competition.

With respect to the acoustic models, the BLSTM improved recognition rate over the DNN. However, as explained when describing the acoustic model training, every acoustic model depends on its previous one. Even if better modeling can lower the error rate, it is interesting to have a robust GMM-HMM initial model.

As for language models, the transformer model yielded the lowest perplexity with respect to the three basic developed language models on the development set, but it was still higher than the interpolated language models. However, when applied to recognition and after having been optimized for the development set, both the transformer and the interpolated language models obtained the best WER overall.

The system is not very tolerant to noise, which can be seen by looking at Table 6.4. This could be explained by the fact that it was mostly trained with noise-free acoustic samples, and no audio filtering step was included in the acoustic data preprocessing.

Moreover, the development error rate reported on Figure 4.4 was certainly misleading: the fact that the BLSTM achieved more than a 50% FER on the development set could have led to believe that the system would perform poorly on the test set, when in fact it was not the case. As explained earlier, background noise in the development set may have been involved in this high FER.

It is interesting to see the effect of changes in the domain. Both the transcriptions from the Tundra corpus as well as the transcriptions from the PolEval 2019 contest were used when testing the language models. However, since the Tundra dataset belongs to a narrative domain, it was harder for the language model to figure out the next word even if newer, better modeling architectures were introduced.

These changes in the domain were reflected in the WER: the system achieved the second best error rates reported on the PolEval 2019 dataset, pulling ahead of Google's speech-to-text system by 3.1%. However, the results were not as great on the Tundra dataset, falling behind Google's speech-to-text system by 14.2%. Considering the test set as a whole, Google's speech-to-text system outperformed the system developed in this work by 10.1%.

Examining the above points, two main takeaways can be extracted:

1. An ASR system is usually applied to a single domain: politics, gaming, news... By choosing different data sources, each with their own standards (for instance, the PELCRA corpora family usually represents hesitation as "yy", "mm" or "ee"), the model was not able to specialize in any domain. Instead, it became an ASR system of general use. This has proven to be a difficult task, which should be addressed with more available data.

2. The set partitioning should take into account the data domain. This point is related to the previous one. The PELCRA EMI, chosen as the development set of the trained models, is a conversational corpus. However, the addition of narrative data on both the training and the development language model sets could have helped reducing error rate on the Tundra corpus. Choosing the right data domain ultimately depends on whether the system is going to be used as a general-purpose system or as a system focused in a certain domain.

This introductory work to the field of ASR leaves many doors open for improvement. Some suggestions are the following:

1. Study the performance of the system with respect to the WIP parameter in combination with the GSF parameter. Due to time constraints, these combinations could not be studied, and a value of $WIP = 0$ was considered in every recognition when experimenting with the GSF. However, a parallel exploration of GSF and WIP could have further benefited accuracy rate.

2. Better define and classify the training, development, and test sets. As it was seen when evaluating the system, it achieved great error rates on political speeches. To further improve the accuracy rate in this domain, one could exclusively use this kind of audios when training the system, thus probably improving accuracy on politics-related speeches.

3. Study the behavior of using more (or less) Gaussians on the GMM-HMM training of the acoustic model. Alignments could be performed with several obtained GMM-HMM models to check the validation error with respect to more advanced model training (DNN and BLSTM training). Furthermore, a 64-Gaussian model could be trained to compare the performance of the obtained models against the 32-Gaussian model used in this work.

4. Explore the topology parameters of the developed networks. For instance, adding more hidden layers or more units per layer may benefit the final model.

5. Train a transformer acoustic model with the data examined in this work (preferably, with data focused on political speeches). Section 2.4.2 hinted that the transformer architecture has become the next state-of-the-art architecture for acoustic modeling. By exploiting this novel architecture, lower error rates could be obtained.

6. Find more data to train both the acoustic and the language models with. As it could be seen when evaluating the system, if a general-purpose system is going to be built, it will need much more data than 225 hours of speech.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Mikhail Arkhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin. Tuning multilingual transformers for language-specific named entity recognition. In *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pages 89–93, Florence, Italy, August 2019. Association for Computational Linguistics.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[4] Pau Baquero Arnal. Comparación de las herramientas informáticas tlk y kaldi para el desarrollo de sistemas de reconocimiento del habla en catalán/valenciano, 2016.

[5] 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.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003.

[7] Vladimír Benko. Aranea: Yet Another Family of (Comparable) Web Corpora. In *TSD*, pages 247–254.

[8] Vladimír Benko. Compatible sketch grammars for comparable corpora. In Andrea Abel, Chiara Vettori, and Natascia Ralli, editors, *Proceedings of the 16th EURALEX International Congress*, pages 417–430, Bolzano, Italy, jul 2014. EURAC research.

[9] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434 – 451, 2008.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[11] Xie Chen, Xunying Liu, Yanmin Qian, M.J.F. Gales, and Philip Woodland. Cued-rnnlm — an open-source toolkit for efficient training and evaluation of recurrent neural network language models. pages 6000–6004, March 2016.

[12] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *NIPS*, 2015.

[13] Sławomir Dadas, MichałPerełkiewicz, and RafałPoświata. Pre-training polish transformer-based language models at scale, June 2020.

[14] 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.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[16] 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.

[17] Sascha Frühholz and Pascal Belin. *The Oxford Handbook of Voice Perception*. Oxford University Press, December 2018.

[18] 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.

[19] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[21] William Y. Huang and Richard P. Lippmann. Neural net and traditional classifiers. In D. Z. Anderson, editor, *Neural Information Processing Systems*, pages 387–396. American Institute of Physics, 1988.

[22] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon, and Raj Reddy. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, USA, 1st edition, 2001.

[23] M. Y. Hwang and X. Huang. Subphonetic modeling with markov states-senone. In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 33–36 vol.1, 1992.

[24] Javier Iranzo Sánchez. A comparative study of neural machine translation frameworks for the automatic translation of open data resources, 2018.

[25] 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.

[26] Danijel Koržinek, Krzysztof Marasek, Lukasz Brocki, and Krzysztof Wolk. Polish read speech corpus for speech tools and services. *CoRR*, abs/1706.00245, 2017.

[27] Shiyu Liang and R. Srikant. Why deep neural networks? *CoRR*, abs/1610.04161, 2016.

[28] Richard P. Lippmann. Speech recognition by machines and humans. *Speech Commun.*, 22(1):1–15, July 1997.

[29] Farhana Ferdousi Liza and Marek Grzes. Improving language modelling with noise-contrastive estimation. *CoRR*, abs/1709.07758, 2017.

[30] Krzysztof Marasek, Lukasz Brocki, Danijel Korzinek, Krzysztof Wolk, and Ryszard Gubrynowicz. Spoken language translation for polish. *CoRR*, abs/1511.07788, 2015.

[31] Krzysztof Marasek, Danijel Koržinek, and Łukasz Brocki. System for automatic transcription of sessions of the polish senate. *Archives of Acoustics*, 39(4), 2014.

[32] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Coherent dialogue with attention-based language models. In *AAAI*, 2017.

[33] Kinfe Tadesse Mengistu and Frank Rudzicz. Comparing humans and automatic speech recognition systems in recognizing dysarthric speech. In Cory Butz and Pawan Lingras, editors, *Advances in Artificial Intelligence*, pages 291–300, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

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

[35] Roger K. Moore. A comparison of the data requirements of automatic speech recognition systems and human listeners. In *INTERSPEECH*, 2003.

[36] Khaled Necibi, Halima Bahi, and Toufik Sari. *Speech Disorders Recognition using Speech Analysis*, chapter 24, pages 494–507. IGI Global, April 2012.

[37] Maciej Ogrodniczuk. Polish Parliamentary Corpus. In Darja Fišer, Maria Eskevich, and Franciska de Jong, editors, *Proceedings of the LREC 2018 Workshop* ParlaCLARIN: Creating and Using Parliamentary Corpora, pages 15–19, Paris, France, 2018. European Language Resources Association (ELRA).

[38] 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.

[39] Piotr Pęzik. Increasing the accessibility of time-aligned speech corpora with spokes mix. In *Proceedings of the Eleventh national Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

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

[41] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

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

[43] Piotr Rybak, Robert Mroczkowski, Janusz Tracz, and Ireneusz Gawlik. Klej: Comprehensive benchmark for polish language understanding, May 2020.

[44] Pavel Rychlý. Manatee/bonito - a modular corpus manager. In *1st Workshop on Recent Advances in Slavonic Natural Language Processing*, pages 65–70, Brno, 2007. Masarykova univerzita.

[45] George Saon, Gakuto Kurata, Tom Sercu, Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis, Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn-Li Lim, Bergul Roomi, and Phil Hall. English conversational telephone speech recognition by humans and machines. pages 132–136, August 2017.

[46] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[47] Holger Schwenk. Continuous space language models. *Computer Speech and Language*, 21(3):492 – 518, 2007.

[48] M. L. Seltzer, D. Yu, and Y. Wang. An investigation of deep neural networks for noise robust speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7398–7402, 2013.

[49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[50] Adriana Stan, Oliver Watts, Y. Mamiya, Mircea Giurgiu, R. Clark, and Junichi Yamagishi. Tundra: A multilingual corpus of found data for tts research created with light supervision. In *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France*, pages 2331–2335, August 2013.

[51] Andreas Stolcke. Srilm - an extensible language modeling toolkit. In *INTERSPEECH*, 2002.

[52] Andreas Stolcke and Jasha Droppo. Comparing human and machine errors in conversational speech transcription. In *Proc. Interspeech*, pages 137–141. ISCA - International Speech Communication Association, August 2017.

[53] Carlos Vaquero, Oscar Saz, Eduardo Lleida, José Marcos, and César Canalís. Vocaliza: An application for computer-aided speech therapy in spanish language. *Proceedings of IV Jornadas en Tecnología del Habla*, November 2006.

[54] 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.

[55] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[56] Chenguang Wang, Mu Li, and Alexander J. Smola. Language models with transformers. *CoRR*, abs/1904.09408, 2019.

[57] 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.

[58] K. Wołk, A. Wołk, and K. Marasek. Big data language model of contemporary polish. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 389–395, 2017.

[59] Dong Yu and Li Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer Publishing Company, Incorporated, 2014.

[60] M. Zimmermann and H. Bunke. Optimizing the integration of a statistical language model in hmm based offline handwritten text recognition. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 541–544 Vol.2, 2004.