TÉCNICO LISBOA

TRIBUS: An end-to-end automatic speech recognition system for European Portuguese

Carlos Manuel Ferreira Carvalho

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Alberto Abad Gareta
Doutor Luís Landeiro Ribeiro

Examination Committee

Chairperson: Prof. Francisco António Chaves Saraiva de Melo
Supervisor: Prof. Alberto Abad Gareta
Member of the Committee: Prof. Isabel Maria Martins Trancoso

January 2021

Dedicated to my mom and dad.

Acknowledgments

In no particular order, I would like to thank Alan Turing and all creators of the movie ”The Imitation Game” from 2014, for introducing me to the field of artificial intelligence, in 2015, which led me to choose computer engineering after spending one year in material engineering. I want to thank professor Alberto Abad that believed in me, and for all the guidance and advice throughout these months. Also, I want to thank Luís Landeiro, actual CEO of the company PDMFC [1], to create the opportunity that made this work possible.

I would also like to thank Thomas Rolland and Pedro Esteves for all the help and presence in all Monday reunions with professor Alberto.

To my friends, I would like to thank for all the support and especially for being present throughout my whole degree: the ones from middle school and before, the ones from when I have been in material engineering, in the first year of university and the ones from computer engineering.

To Júlia, I would like to thank you for everything, since I do not have any better words to describe it.

Finally, I am very thankful and indebted to all my family for all support and confidence they have placed in me. In particular, I would like to thank my dad, my mother, and my sister that supported me unconditionally in all situations, which inspired me to become a better version of myself and always learn and evolve constantly.

[1] https://www.pdmfc.com

Resumo

Os sistemas end-to-end em tradução de fala para texto surgiram como modelos competitivos para os modelos tradicionais, baseados em HMMs. Contudo, a maioria dos sistemas end-to-end para tradução de fala para texto não são fáceis de reproduzir, maioritariamente porque é necessário dispor um grande conjunto de dados e poder computacional. Consequentemente, existem poucos resultados para línguas em que os dados são limitados, como por exemplo: Português Europeu. Neste trabalho vamos apresentar um conjunto de experiências feitas com o objetivo de criar os melhores sistemas end-to-end, em Português Europeu, com recurso a poucos dados. O sistema proposto, chamado TRIBUS, é um sistema híbrido que combina CTC e Attention. Os dados utilizados contêm fala lida, fala telefónica e fala de notícias. Para avaliarmos o sistema end-to-end, treinámos um modelo baseado em HMMs no mesmo conjunto de dados. Os resultados experimentais mostram que o modelo TRIBUS consegue obter um erro de caracter de 8.40%, no conjunto de teste do domínio de fala de noticiário, que é comparável com os 4.33%, obtidos pelo modelo base de comparação, no mesmo conjunto de teste. Para concluir, propusemos também um novo método para treinar sistemas CTC, através do auxílio de um mecanismo de memória. Este novo sistema funciona melhor do que apenas usar CTC.

Palavras-chave: tradução de som para texto, end-to-end, modelos híbridos com CTC e attention, poucos recursos, modelos baseados em memória.

Abstract

End-to-end automatic speech recognition (ASR) approaches have emerged as a competitive alternative to traditional HMM-based ASR systems. Unfortunately, most end-to-end ASR systems are not easily reproduced since they require vast amounts of data and computational resources that are only available for a reduced set of companies and labs worldwide. Consequently, the performance of these systems is not very well known for low resource languages to the best of our knowledge. European Portuguese is one of those languages. In this work, we present a set of experiments to train and assess some of the most current successful end-to-end ASR approaches for European Portuguese. The proposed system, named TRIBUS, is a hybrid CTC-attention end-to-end ASR combining data from three different domains: read speech, broadcast news and telephone speech. For comparison purposes, we also train a state-of-the-art HMM-based baseline on the same data. Experimental results show that TRIBUS achieves 8.40% character error rate (CER) on the broadcast news test set without the need of a language model, which is comparable to the strong baseline result, 4.33% CER, on the same set using an in-domain language model. We consider this result quite promising, especially for highly unpredictable vocabulary ASR applications. Finally, and more notably, a novel way of training CTC-based models using a memory-based approach, that performs better than only using CTC alone, was developed.

Keywords: automatic speech recognition, end-to-end, hybrid CTC-attention, low resources, memory-based approaches.

Contents

List of Tables

List of Figures

Acronyms

AM Acoustic model.

ANN Artificial neural network.

ASR Automatic speech recognition.

AT Adversarial training.

BN Broadcast news.

BPTT Back-propagation through time.

CART Classification and regression tree.

CER Character error rate.

CMU Carnegie Mellon university.

CMVN Cepstral mean and variance normalization.

CNN Convolutional neural network.

CTC Connectionist temporal classification.

DARPA Defense Advanced Research Projects Agency.

DCT Discrete cosine transformation.

DNN Deep neural network.

EM Expectation-maximization.

FC Fully connected layer.

FST Finite state transducer.

G2P Grapheme to phoneme.

GANs Generative adversarial networks.

GD Gradient descent.

GMM Gaussian mixture model.

HMM Hidden markov model.

IPA International phonetic alphabet.

LDA Linear discriminant analysis.

LM Language model.

LSTM Long-short term memory.

LVCSR Large vocabulary continuous speech recognition.

MFCC Mel frequency cepstral coefficients.

MLLT Maximum likelihood linear transform.

MLP Multilayer perceptron.

NTM Neural Turing machine.

OOVs Out of vocabulary words.

PER Phone error rate.

ReLU Rectified linear unit.

RNN-T Recurrent neural network transducer.

RNN Recurrent neural network.

SAT Speaker adaptive training.

SGD Stochastic gradient descent.

SLM Statistical language modeling.

STS Speech to speech.

SVD Singular value decomposition.

TDNN-F Time delay neural network factorized.

TDNN Time delay neural network.

WER Word error rate.

WFST Weighted finite state transducer.

WSJ Wall street journal.

Chapter 1

Introduction

Speech recognition technology is submerged in our society more than ever. Products like Siri, Cortana, Google Now, Amazon Echo Alexa are part of our every day lives. This high tech translates into a big number of applications that guides our society to a better living. Some of the applications are *in car systems*, e.g., to initiate a phone call or to ask for directions; *in hands-free computing* that are useful for both able and disabled users; *healthcare* support [1] and *speech-to-speech* translation systems [2]. It is also interesting to note that giving the speech as an input to a device is three times faster than typing in the keyboard [3].

This work is intended to create a speech recognition system for European Portuguese, by using some of the latest technologies present in the automatic speech recognition (ASR) community. Included in this chapter are the motivations for realising this project, the objectives and a brief overview of the document structure.

1.1 Motivation

Traditional speech recognition systems rely on sophisticated pipelines, manually created by computer scientists and engineers. Since 1980, these systems were composed by Hidden Markov models (HMMs), which model the probability of going from one acoustic state (generally a triphone) to another, and Gaussian mixture models (GMMs) that model the probability of occurrence of that particular acoustic state.

The combination of the HMMs and GMMs constitute what is commonly known as the acoustic model of the speech recognition system, which is eventually combined with a language model and a pronunciation dictionary to then generate the text transcriptions, through a decoding process. The creation of these modules for HMM-GMM systems is favourable when few data is available to train since they already contain some speech and language knowledge. Nonetheless, the creation of these individual and complex systems can limit the potential performance of speech recognition, mainly because they can be poor approximation models of reality, e.g., forcing an algorithm to use a phonetic representation can limit the speech system performance [4].

In 2006, artificial neural networks (ANNs) resurged with a new name: deep learning. This occurred

mainly because it started to be possible to train networks with more layers, using a new greedy layer-wise unsupervised pretraining technique [5]. Today, we know that unsupervised pretraining is not required to train deep neural networks as a result of new, among many, initialization strategies, activation functions (e.g., ReLu [6]) and adaptive optimization algorithms (e.g., Adam [7]). In the beginning, state-of-the-art was achieved in ASR by just replacing the GMMs with deep feedforward neural networks (DNNs), since the latter are better in modelling data that lie on or near a non-linear manifold, as opposed to the former [8].

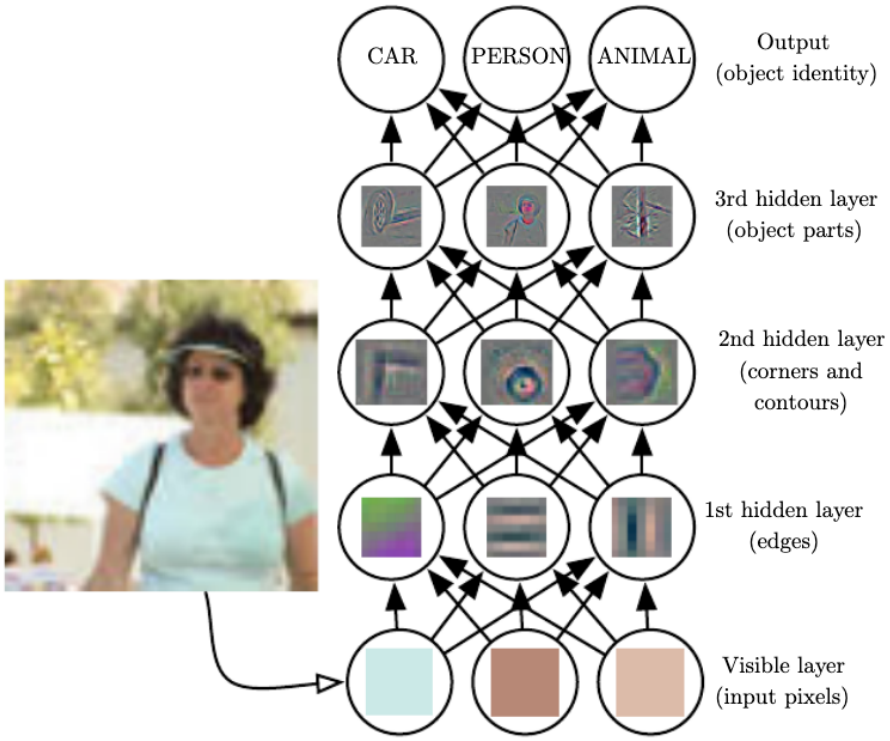


Figure 1.1: Deep learning breaks down the complex mapping of the input image to the respective class by letting the first layers to learn simpler features, e.g., edges, and later layers to represent more complex features, i.e., object parts. In theory, this learning procedure also applies to speech recognition in the sense that waves can break down to have simpler representations (e.g., phonemes) at lower level layers, and more complex representations in higher-level layers, like words and sentences. Source from [9].

These conventional ASR architectures, mentioned above, have limitations, mainly because they are based on HMMs and contain various sub-models that deal with separate acoustic, pronunciation and language models. First, the creation of all different models requires expert knowledge and is time-consuming. They are also trained with different goals from the final evaluation metric, e.g., word error rate (WER). To create a state-of-the-art HMM-DNN system, it is first required to train an HMM-GMM system to obtain phonetic alignments. Moreover, since the decoding stage is performed by integrating all modules with finite-state transducers (FSTs), creating and implementing these well-optimized transducers is very complex. Finally, the HMM systems and n-gram language models make conditional independence assumptions, whereas real speech does not follow those strict assumptions.

A demanding task that is still ongoing is to fully replace this module-based architecture and replace

the entire traditional pipeline with a fully differentiable DNN architecture to eliminate the above-outlined issues. This is possible, since deep learning technology enables the machine to create more abstract concepts out of simpler ones, as depicted in figure 1.1. Today, the main drawback of this technology, named end-to-end, is that it requires much more data (usually more than a thousand hours) and more computational power [10, 11], when compared to HMM-based systems. Notwithstanding, in the presence of big data resources, these systems achieve state-of-the-art results when compared to the HMM-based systems [12, 13]. Consequently, most modern ASR products provided by big companies like Amazon, Google and Apple, are based on end-to-end.

In 2006, it was proposed connectionist temporal classification (CTC) [14], the first technique that was closer to an end-to-end system. CTC allows training end-to-end systems without requiring alignments between input features and output labels. It is still not an end-to-end system because CTC does not model the interdependencies between the outputs, which then requires a language model. To solve this CTC assumption problem, recurrent neural network (RNN)-transducer, or RNN-T, was proposed in 2012 [15], where the CTC is jointly trained with an RNN that learns linguistic information. The other main technique for training end-to-end ASR, proposed in [16, 17], is the attention end-to-end encoder-decoder system. This system is divided into an encoder, which acts as the acoustic model, an attention layer that learns the alignments between input and output. At last, the decoder acts as the language model. Usually, the outputs of these systems are characters or subwords. Beyond these main architectures for end-to-end ASR systems, there are many ways to create variants of it, by using adversarial training [18] and memory-based approaches [19, 20].

Numerous efforts have been created to develop speech recognition systems for the English language [8, 10, 16, 17], and for this reason many available English data sets, e.g., TIMIT [21], Librispeech [22] and Wall Street Journal (WSJ) [23], were created to be used as baselines for comparison between different kinds of ASR systems. For European Portuguese, there is no evidence of work in the literature of end-to-end speech recognition. This is mostly due to the lack of publicly available large-scale speech data resources, either paid or for free on websites like VoxForge[1].

1.2 Objectives

Our prime goals for this work will consist on:

- Understanding the main advantages and disadvantages of today end-to-end systems by conducting an extensive literature review.
- Studying and attaining more insights into the advantages and disadvantages of the two main end-to-end systems that exist, i.e., CTC-based and attention-based systems.
- Selecting the data that will be used for all experiments.
- Developing a strong HMM-DNN baseline system for European Portuguese.

[1] http://www.voxforge.org/home

- Creating the best possible end-to-end system for European Portuguese.
- Comparing the recognition performance of baseline HMM-DNN with the end-to-end system for European Portuguese.
- Investigating the use of auxiliary iVectors and adversarial training to create better speaker invariant representations.
- Proposing a novel CTC-based ASR system combined with memory-based approaches to improve the current state-of-the-art.

After achieving these main objectives, it will be possible to answer the two main research questions in this work. First, we want to know how well end-to-end systems can perform when compared to HMM-DNN systems for the particular case of European Portuguese with limited resources. Second, we also aim to investigate and propose novel ways for either creating improved speaker-invariant representations or incorporating memory modules into current architectures that can improve the current state-of-the-art.

1.3 Thesis outline

This thesis is structured into six chapters. Chapter 1 corresponds to the introduction. Next, chapter 2 contains relevant background concepts that help to understand the work created. More specifically, it mentions some introductory concepts about HMMs, some deep learning algorithms and an introduction to the field of ASR. Chapter 3, will be dedicated to explaining all details about every corpus used in the experiments, such as the creation of the TRIBUS corpus. Chapters 4 and 5 correspond to the creation and training of HMM-based and end-to-end ASR systems for European Portuguese, respectively. Both at the beginning, they include related work about the evolution of respective state-of-the-art ASR systems, followed by experiments, results and a discussion section. More importantly, the experiments to create better speaker invariant representations and propose the novel CTC-based ASR system combined with memory-based approaches are detailed in chapter 5. Finally, chapter 6 contains a summary of the main conclusions for this work and indicates several promising directions for future research, more specifically with end-to-end ASR systems for low resources.

Chapter 2

Background

This chapter describes fundamental knowledge that is mandatory to understand HMM-based and end-to-end ASR systems. In section 2.1 a brief description of HMMs and algorithms associated with them are described and in section 2.2 all tools required to understand how end-to-end ASR systems work are explained. Finally, in section 2.3, a small overview of the evolution of ASR and the main goal and challenges for the field will be introduced.

2.1 Hidden Markov models

Hidden Markov Models (HMMs), named after the Russian mathematician Andrey Andreyevich Markov, are statistical models that capture hidden information from observable sequential symbols and were first applied to speech recognition in 1975, by Baker and his wife Janet [24].

HMMs rely on Markov Chains, that can be simply described as graphs with transition probabilities from one given state i to another state j, denoted as $p_{i,j}$. If the probability of being in a given state depends only on the previous one, instead of relying on all past states, we are in the presence of a Markov Process. A Markov Chain with this property, is also known as a first order Markov Chain. On the other hand, if being in a state depends on the previously two states, we are in the presence of a second order Markov Chain, and so on.

Formally an HMM can be defined as a tuple $\lambda = (\mathbb{S}, \mathbb{Y}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\mu})$, where:

- $\mathbb{S} = \{s_1, s_2, ..., s_n\}$ is a set of $|\mathbb{S}| = n$ hidden states that are assumed to be a first order Markov Process. The state at time t is denoted by $q_t = s_j \in \mathbb{S}$;

- $\mathbb{Y} = \{y_1, y_2, ..., y_m\}$ is a set of $|\mathbb{Y}| = m$ observable states. The observation state at time t is denoted by $o_t = y_j \in \mathbb{Y}$;

- $\boldsymbol{A} = \{A_{i,j}\}$ is a transition probability matrix of size $n \times n$, where

$$\{A_{i,j}\} = P(q_{t+1} = s_j | q_t = s_i); \tag{2.1}$$

- $\boldsymbol{B} = \{b_j(k)\}$ is the corresponding observation probabilities matrix of size $n \times m$, where

$$b_j(k) = P(o_t = y_k | q_t = s_j), \tag{2.2}$$

  and $b_j(.)$ is some probability distribution function, e.g., Multivariate Gaussian distribution;

- $\boldsymbol{\mu} = \{\boldsymbol{\mu}_i\}$ is the initial probability distribution matrix of size $1 \times n$, where

$$\boldsymbol{\mu}_i = P(q_1 = s_i); \tag{2.3}$$

All these matrices are row stochastic, meaning that all entries are probabilities and every row adds up to 1.

Rabiner [25] introduced the concept that HMMs should be characterized by three fundamental problems - *Likelihood* problem, *Decoding* problem and *Learning* problem - described, respectively, in the following sections.

Problem 1: Likelihood

The main goal of the first problem is to compute the likelihood $P(\boldsymbol{o}|\lambda)$ of an observed sequence $\boldsymbol{o} = [o_1, o_2, ..., o_T]$, given the model λ. Let $\boldsymbol{q} = [q_1, q_2, ..., q_T]$, then we have

$$P(\boldsymbol{o}|\boldsymbol{q}, \lambda) = \prod_{t=1}^{T} P(o_t | q_t, \lambda). \tag{2.4}$$

Since we do not know the hidden sequence $\boldsymbol{q}$, we would have to compute $P(\boldsymbol{o}, \boldsymbol{q}|\lambda)$ for every possible sequence and then sum up all to get $P(\boldsymbol{o}|\lambda)$. This approach is exponential in T. A better approach is to use *forward-algorithm*, which calculates the probability of the partial observation $o_1 o_2 ... o_t$ and state s_i until time t. Let $\alpha_t(j) = P(o_1 o_2 ... o_t, q_t = s_j | \lambda)$, then we can compute forward-algorithm recursively as:

- Initialization: for i = 1,...,n

$$\alpha_1(i) = \mu_i b_i(o_1). \tag{2.5}$$

- Induction: for t = 1,...,T-1, and for j = 1,...,n

$$\alpha_{t+1}(j) = b_j(o_{t+1}) \sum_{i=1}^{n} \alpha_t(i) A_{i,j} \tag{2.6}$$

- Evaluating the probability:

$$P(\boldsymbol{o}|\lambda) = \sum_{i=1}^{n} \alpha_T(i) \tag{2.7}$$

Problem 2: Decoding

The aim of the decoding problem is to find the hidden state sequence that was most likely to produce a sequence of observations $\boldsymbol{o}$, given a model λ. *Viterbi* algorithm is the one that finds that most probable hidden state sequence. It is also a dynamic programming algorithm, because it maintains the highest probable path at each possible state. Viterbi algorithm updates each new value based on the last ones already computed:

- Initialization: for j = 1,...,n

$$v_1(j) = \mu_j b_j(o_1). \tag{2.8}$$

- Induction: for t = 2,...,T, and for j = 1,...,n

$$v_t(j) = max\{v_{t-1}(i) A_{i,j} b_j(o_t)\}. \tag{2.9}$$

At the end, the best path is discovered by using a backtrack search through the computed values.

Problem 3: Learning

The third problem, learning the parameters of an HMM given a sequence observation $\boldsymbol{o}$ that maximizes $P(\boldsymbol{o}|\lambda)$, is solved by using *Baum-Welch* algorithm, which is a special case of the expectation-maximization (EM) algorithm [26]. The main purpose is to tune the parameters of the HMM, namely $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{\mu}$, so that the likelihood of generating $\boldsymbol{o}$ by model λ is maximized.

Before diving in into the algorithm it will be first described *forward-backward* algorithm which is used to find the most likely state for any point in time. The first part, which corresponds to the forward algorithm described above, goes forward in time by computing the probability of ending in a particular state, $P(q_t = s_j|o_1, ..., o_t)$. The second part goes backward in time, by computing the probability of observing the remaining observations given any starting point t, $P(o_{t+1}, ..., o_T|q_t = s_j)$. Backward algorithm is similar to the forward algorithm, where the only difference is that, it starts at the end and then go backwards until it reaches the beginning. Let $\beta_t(j) = P(o_{t+1}, ..., o_T|q_t = s_j, \lambda)$, then we can compute backward algorithm recursively as:

- Initialization: for i = 1,...,n

$$\beta_T(i) = 1. \tag{2.10}$$

- Induction: for t = T-1,...,1 and for i = 1,...,n

$$\beta_t(i) = \sum_{j=1}^{n} A_{i,j} b_j(o_{t+1}) \beta_{t+1}(j) \tag{2.11}$$

Then we can define, for t = 1,...,T and i = 1,...,n,

$$\gamma_t(i) = P(q_t = s_i|\boldsymbol{o}, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(\boldsymbol{o}|\lambda)}, \tag{2.12}$$

which provides the probability of being in each state at time step t. The most likely state at time step t is the one where $\gamma_t(i)$ is maximum. It is important to note that the vector sequence, of individually probable states, could not be the most probable sequence for a given $\boldsymbol{o}$ and model λ.

In Baum-Welch algorithm, forward-backward algorithm, explained above, is first computed, and finally, an update is done to λ parameters. These two steps are executed as many times as needed until convergence is reached. Before the update step, it is computed $\gamma_t(i)$, explained already when computing the forward-backward algorithm, and $\xi_t(i,j)$, which is the probability of being in state s_i at time t and in state s_j at time step t+1.

$$\xi_t(i,j) = \frac{\alpha_t(i)A_{i,j}b_j(o_{t+1})\beta_{t+1}(j)}{P(\boldsymbol{o}|\lambda)} \tag{2.13}$$

When updating parameters of λ, the following procedure is executed:

- Update $\boldsymbol{\mu}$: for i = 1,...,n

$$\mu_i = \gamma_1(i) \tag{2.14}$$

- Update $\boldsymbol{A}$: for i = 1,...,n and for j = 1,...,n

$$A_{i,j} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)} \tag{2.15}$$

- Update $\boldsymbol{B}$: for j = 1,...,n and for k = 1,...,m

$$b_j(k) = \frac{\sum_{t=1}^{T}\xi_t(i,j)}{\sum_{t=1}^{T}\gamma_t(i)} \tag{2.16}$$

The whole process of Baum-Welch algorithm can be described as: (I) in the beginning λ parameters are initialized, e.g., randomly; (II) next, it is computed $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$ and $\xi_t(i,j)$; (III) finally, λ parameters are re-estimated, and while $P(\boldsymbol{o}|\lambda)$ increases we repeat all process from step (II).

It is important to note that the algorithm does not guarantee convergence to a global maximum.

2.2 Deep learning

Deep learning is an approach to artificial intelligence, more specifically to machine learning, that has gone through several different connotations throughout the years, as discussed in [9]. Currently, the connotation ”deep learning” emphasises the fact that it is possible to learn deeper neural networks, something not conceivable until 2006, due to new techniques for optimizing the training on deeper networks, larger datasets mainly because of Moore law applied to data storage capacity and the fast growth of internet usage and, at last, more powerful computer resources like GPUs.

The foundation block of deep learning is an *artificial neural network* (ANN), also named *deep feedforward network*, as mentioned in section 2.2.1.

ANNs are computational models inspired by neuroscience, that date to the 1940s [27]. Despite of this motivation, modern research in neural networks does not aim to mimic the brain. Instead, with the auxiliary of mathematical frameworks and engineering, the goal is to create function approximation algorithms that achieve statistical generalization of the known data.

Formally a neural network is composed by neurons, that are organized in layers, each with one or more neurons. ANNs can be described as graphs, where neurons are nodes and the connections between them are edges. Commonly there is an input layer, an output layer and an arbitrary number of hidden layers that is subject to the network architecture. The simplest version of a network that contains only one neuron is named perceptron [28]. Each node applies a previously chosen activation function, φ, to an affine transformation of its input vector, $\boldsymbol{x} = (x_0, x_1, ..., x_N)^T$, returning a single output $\boldsymbol{h}$:

$$\boldsymbol{h} = \phi(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}) \tag{2.17}$$

The weights $\boldsymbol{W} = (w_0, w_1, ..., w_N)^T$, and bias $\boldsymbol{b}$, are the parameters, $\boldsymbol{\theta}$, that must be learned, in order to find the best mapping function, $f(\boldsymbol{x}; \boldsymbol{\theta})$, that approximates to the real one, f^*.

In the following sections we will describe more in depth deep feedforward networks associated with some optimizers and regularizers commonly used, convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Finally, in section 2.2.4, a brief description of the best practices in deep learning, some created and taught by Andrew Ng and extensively used for this project will be mentioned [1].

2.2.1 Deep feedforward networks

A feedforward neural network, also known as a *multilayer perceptron* (MLP), relates with a directed acyclic graph, where information flows through only one direction. Usually this flow starts at the input layer, then goes through the hidden layers and finally to the output layer. If there are feedback connections between nodes, the network becomes a recurrent neural network, as presented in section 2.2.3.

In a feedforward network the output y is a result of a composition of functions. For instance, if we have four functions f^1, f^2, f^3 and f^4, they will be arranged like $f(\boldsymbol{x}) = (f^4 \circ f^3 \circ f^2 \circ f^1)(\boldsymbol{x})$, where f^1 is the first layer, f^2 and f^3 are second and third layers, respectively, and so on. Assuming that all nodes in the chain have linear activation functions the equation can be simplified to $f(\boldsymbol{x}) = \boldsymbol{W}^{'T}\boldsymbol{x} + \boldsymbol{b}^{'}$. Thus a feedforward neural network made of only linear nodes, that can only "learn" from linearly separable data, has limitations in creating statistical generalization to non-linear data, e.g., XOR problem [29]. By including non-linear activation functions in at least one hidden layer, a feedforward network provides a *universal approximation framework* [30, 31]. In simple terms, feedforward networks can be used to approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$ to any desired degree of accuracy. The main problems with the universal theorem are that the number of hidden units may be too large and the resulting learned model may fail to generalize to unseen data.

[1] `https://www.deeplearning.ai/deep-learning-specialization/`

As mentioned above, activation functions are applied on top of an affine transformation, usually in an element-wise way. Below some of the most important activation functions will be described.

Activation Functions

The *sigmoid* function, observed in equation 2.18, receives a value x and "squashes" it to a value between 0 and 1. One limitation of the sigmoid activation function is that it saturates to a very high value when x is very high and saturates to a low value when x is a low value, thus the gradients at these regions are very close to 0. This problem, known as the "vanishing gradient" makes gradient-based optimization much more difficult [32]. Sigmoid is commonly applied in the output layer to represent a probability distribution over a discrete binary variable.

$$\phi(x) = \sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.18}$$

The *hyperbolic tangent* activation function, also named *tanh*, is intrinsically related to the sigmoid because

$$\phi(x) = tanh(x) = 2\sigma(2x) - 1. \tag{2.19}$$

It "squashes" x into a value between -1 and 1 and has the same problem of the sigmoid activation function, nevertheless, since it is similar to the identity function in the way that $tanh(0) = 0$, in contrary to the sigmoid where $\sigma(0) = 1/2$, it can optimize better because it has "stronger" gradients, therefore it can help to converge faster.

The *rectified linear unit* (ReLU) [6] computes the function

$$\phi(x) = max\{0, x\}, \tag{2.20}$$

which is much more efficient to optimize when compared to the sigmoid and tanh mentioned above, due to the linear non-saturating form. Another advantage is that it performs "cheaper" computations compared to the expensive exponentials of the sigmoid and tanh. As a result of these advantages, ReLU is the most widely used activation function at the level of the hidden layers. One drawback of ReLUs, named "dying ReLU", happens when the activation is equal to zero, which means that the gradient will also be zero and consequently, the ReLU unity will stay at zero indefinitely. There are other generalizations of ReLU that address this problem, so that gradient exists everywhere, e.g., leaky ReLU [33] or parametric ReLU [34].

Another important activation function is named *softmax*, an extension of the sigmoid which can only represent a discrete binary variable as mentioned above. Softmax functions can represent probability distributions over a discrete variable with n values. Very often, softmax functions are used at the output of networks to represent the probability distribution over n different labels. Formally, if $\boldsymbol{z} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$ and

$z_i = P(y = i|\boldsymbol{x})$, the softmax activation function can be defined as:

$$\phi(z_i) = softmax(\boldsymbol{z})_i = \frac{exp(z_i)}{\sum_j exp(z_j)}. \tag{2.21}$$

A nice property of softmax activation function is the fact that it is continuous and differentiable, thus the name "soft". We can also observe that the probability distribution sums up to 1 and all probabilities are positive.

Now that we know how to compute affine transformations using element-wise activation functions the next section mentions the algorithm that will help to modify the parameters θ, in order to find the best mapping function $f(\boldsymbol{x}; \boldsymbol{\theta})$ which approximates to the real one f^*.

Back-Propagation

Deep feedforward neural networks are commonly trained by using iterative gradient-based optimizers in order to reduce the value of an objective function, $J(\boldsymbol{\theta})$, as low as possible. This objective function is usually named as cost function. Due to the non-linearity of the neural network, most of the cost functions become non-convex, therefore the iterative process does not guarantee convergence at a global minimum, because of the existence of many local minimum and saddle points.

Before learning, two main steps are involved. The first one, called *forward propagation*, consists on propagating $\boldsymbol{x}$ into the input layer, then the information flows forward through the hidden layers until it finally creates an output $\hat{y}$ with a scalar cost $J(\boldsymbol{\theta})$ associated. The second step, called *back-propagation* algorithm [35], works the other way around. The information $J(\boldsymbol{\theta})$ flows backwards through the network to compute the gradient. Finally, this gradient is used by an optimization algorithm, e.g., *gradient descent* (GD), to perform the learning (the adjustments of the parameters, $\boldsymbol{\theta}$, of the neural network). Some of the most common used optimization algorithms will be described in more detail at section 2.2.1.

Back-propagation relies on the chain rule of calculus. If $f = g(h(x))$ and $y = h(x)$, where $g, h : \mathbb{R} \to \mathbb{R}$, then we have

$$\frac{df}{dx} = \frac{df}{dy}\frac{dy}{dx} \tag{2.22}$$

By generalizing beyond the scalar we can denote the same thing in a vector notation. Let $h : \mathbb{R}^m \to \mathbb{R}^n,\ g : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$, then

$$\nabla_{\boldsymbol{x}} f \ = \ \left(\frac{\partial\, \boldsymbol{y}}{\partial\, \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} f \tag{2.23}$$

where $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the $n \times m$ jacobian matrix of h. Thus, back-propagation consists on the multiplication of jacobians with gradients.

Let L be the total number of layers in a fully connected MLP with an input $\boldsymbol{x}$ and a corresponding output y. The affine transformation for layer k can be defined as $\boldsymbol{a}^{(k)} = \boldsymbol{W}^{(k)T}\boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)}$ and when the activation function is applied on top of the affine transformation, $\boldsymbol{a}^{(k)}$, we obtain $\boldsymbol{h}^{(k)} = \phi(\boldsymbol{a}^{(k)})$.

It is important to note that $\boldsymbol{h}^{(0)} = \boldsymbol{x}$. After the forward pass, the output produced can be defined as $\hat{y} = \phi(\boldsymbol{a}^{(k)})$. With this information we can start by applying back-propagation through the network in order to calculate the gradients with respect to the parameters $\boldsymbol{\theta}$ (weights and biases) for each layer.

For instance, for the last layer k, by applying the chain rule to $\nabla_{\boldsymbol{W}^{(k)}} J(\boldsymbol{\theta})$ we get:

$$\nabla_{\boldsymbol{W}^{(k)}} J(\boldsymbol{\theta}) = \nabla_{\hat{y}} J(\boldsymbol{\theta})\, \nabla_{\boldsymbol{a}^{(k)}} \hat{y}\, \nabla_{\boldsymbol{W}^{(k)}} \boldsymbol{a}^{(k)} \tag{2.24}$$

where $\nabla_{\boldsymbol{a}^{(k)}} \hat{y} = \phi'(\boldsymbol{a}^{(k)})$ and $\nabla_{\boldsymbol{W}^{(k)}} \boldsymbol{a}^{(k)} = \boldsymbol{h}^{(k-1)}$. For $\nabla_{\boldsymbol{b}^{(k)}} J$ the same process is applied. Then, after propagating the gradients w.r.t the next lower-level hidden layers activation k-1, $\boldsymbol{g} = \nabla_{\boldsymbol{h}^{(k-1)}} J(\boldsymbol{\theta}) = \nabla_{\hat{\boldsymbol{y}}} J(\boldsymbol{\theta})\, \nabla_{\boldsymbol{a}^{(k)}} \hat{y}\, \nabla_{\boldsymbol{h}^{(k-1)}} \boldsymbol{a}^{(k)}$, where $\nabla_{\boldsymbol{h}^{(k-1)}} \boldsymbol{a}^{(k)} = \boldsymbol{W}^{(k)}$, we get:

$$\nabla_{\boldsymbol{W}^{(k-1)}} J(\boldsymbol{\theta}) = \boldsymbol{g}\, \nabla_{\boldsymbol{a}^{(k-1)}} \boldsymbol{h}^{(k-1)}\, \nabla_{\boldsymbol{W}^{(k-1)}} \boldsymbol{a}^{(k-1)} \tag{2.25}$$

where $\nabla_{\boldsymbol{a}^{(k-1)}} \boldsymbol{h}^{(k-1)} = \phi'(\boldsymbol{a}^{(k-1)})$ and $\nabla_{\boldsymbol{W}^{(k-1)}} \boldsymbol{a}^{(k-1)} = \boldsymbol{h}^{(k-2)}$. This propagation goes on until it is reached the first layer of the feedforward network.

At each iteration, going layer by layer in the back-propagation algorithm, we can update the parameters with their respective gradients by using, for example, GD where $\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(k-1)} - \epsilon \times \nabla_{\boldsymbol{\theta}^{(k)}} J(\boldsymbol{\theta})$ and ϵ is a positive scalar that determines the size of the step. In machine learning literature, ϵ is also called *learning rate*.

In this section it has been described how to compute derivatives by using back-propagation, but it is relevant to know that there are other optimal ways to compute the gradient and more subtleties associated with it [9].

Optimization

Deep learning optimization is focused on finding the parameters $\boldsymbol{\theta}$ that reduce the cost function $J(\boldsymbol{\theta})$ on a *training set*, with the auxiliary of back-propagation, so that in the end it can generalize well to the unseen data, called *test set*. For this reason, it is relevant to note that deep learning optimization differs from pure optimization techniques.

Usually, optimization algorithms for deep learning compute each update to the parameters based on an expected value of the cost function estimated using only a subset of the full data set examples. These subsets, greater than one and smaller than all examples of the data set, are named *mini-batch stochastic* or *stochastic*. One pass in the full training set is named an *epoch*. It is important to note that usually, these mini-batches need to be selected in a random way to increase the effectiveness of the algorithm, as mentioned in [9].

The most basic and widely used optimization algorithm is called *stochastic gradient descent* (SGD), which is a much faster version when compared to gradient descent. At each step, it computes an average of the gradient on a mini-batch of m examples drawn i.i.d from training data as

$$\widehat{\boldsymbol{g}} = \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i} LossFunction(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), f^*(\boldsymbol{x}^{(i)})), \tag{2.26}$$

where *LossFunction* denotes the per-example loss function. Then the parameters can be updated using

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon \times \widehat{\boldsymbol{g}}, \tag{2.27}$$

in order to minimize the error on the test set, also called generalization error.

Learning with SGD can sometimes be slow because of the poor conditioning of the Hessian matrix, which can be improved by using momentum, as observed in figure 2.1.

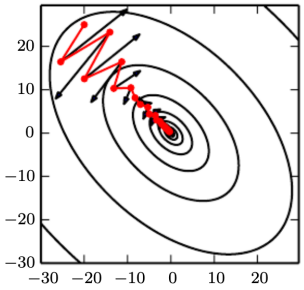


Figure 2.1: The contour lines represent a quadratic loss function with a poorly conditioned hessian matrix. The red path represents the path followed by the momentum learning rule. The black arrow represents the step that SGD would take without momentum. Source from [9].

Briefly, the SGD momentum algorithm accumulates an exponentially decaying moving average of the past gradients, therefore speeding up convergence and helping to escape from local minima:

$$\boldsymbol{v} = \alpha\boldsymbol{v} - \epsilon\nabla_{\boldsymbol{\theta}}\frac{1}{m}\sum_{i} LossFunction(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), f^{*}(\boldsymbol{x}^{(i)})), \tag{2.28}$$

where the hyperparameter $\alpha \in [0, 1)$ determines how quickly the contributions of past gradients exponentially decay. The update rule for the parameters of the network is given by

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{v}. \tag{2.29}$$

It is relevant to note that there are many more sophisticated first-order optimization methods (that use only the first derivative) like Adagrad [36], RMSProp [2] and Adam [7].

Batch normalization [37] is a method of adaptive reparametrization for training very deep models, that standardizes the input of a layer for each mini-batch, working as a regularizer and speeding the optimization procedure. One reason, proposed in [37], for being hard to train deep models is that the distribution of the inputs to deep layers in the network may change after each mini-batch when weights are updated. This problem was coined as ”internal covariate shift”. Despite this belief, there are some suggestions that instead, batch normalization is successful because it smooths and in turn, simplifies the optimization function when training the network [38]. It is named batch normalization because, during training, the input layers are normalized using the mean and standard deviation of the values in the current mini-batch. Let $\boldsymbol{H}$ be a mini-batch of activations for one node of the layer, where the activations

[2] http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

for each example correspond to each row in the matrix. Normalizing $\boldsymbol{H}$ corresponds to computing:

$$\boldsymbol{H}^{'} = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}, \tag{2.30}$$

where $\boldsymbol{\mu}$,

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i} \boldsymbol{H}_{i,:}, \tag{2.31}$$

is a vector containing the mean of each unit and $\boldsymbol{\sigma}$,

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_{i}(\boldsymbol{H} - \boldsymbol{\mu})_i^2}, \tag{2.32}$$

is a vector containing the standard deviation of each unit. δ is usually a small positive value that avoids the denominator of $\boldsymbol{H}^{'}$ to become 0 and $\boldsymbol{H}_{i,:}$ corresponds to row i of matrix $\boldsymbol{H}$. The rest of the network then operates on $\boldsymbol{H}^{'}$.

Normalizing the mean and standard deviation of a unit to 0 and 1, respectively, can reduce the network expressive power to represent a different distribution [9]. For this situation, it is applied γ and $\boldsymbol{\beta}$ to the normalized values, which become learnable values of the network. Instead of now using $\boldsymbol{H}^{'}$, we have:

$$\boldsymbol{\gamma}\boldsymbol{H}^{'} + \boldsymbol{\beta}. \tag{2.33}$$

At test time it is common to replace $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ by averages collected during training, which allows the model to be evaluated in just one example at a time.

Weight initialisation

Training deep learning models with sophisticated optimization algorithms, by nature, is a very complex task. Above that, most of the optimization tasks in deep learning are strongly affected by choice of the initial parameters. The starting point can determine whether the algorithm converges at all, or how quickly it converges and whether it converges to a point with high or a low-cost value. Commonly, the weights are initialized by randomly sampling values from a Gaussian or uniform distribution. A more sophisticated technique is *normalized initialization* [39], commonly named as Xavier initialization:

$$\boldsymbol{W}_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right), \tag{2.34}$$

where m is the number of inputs, and n the number of outputs of the corresponding fully connected layer.

Only the weights have been mentioned, but the bias terms also play an essential role in the initialization of the deep learning model. Very commonly these are set to zero.

It is also worth noting that it is also possible to initialize a supervised model with parameters learned

by an unsupervised model trained on the same input.

Regularization

Throughout training, DNN models usually increase their accuracy on training set while degrading accuracy on test set. This phenomenon, known as overfitting, is solved by applying a set of strategies in order to reduce the test error. These strategies, also known as *regularization*, can impose conditions on the model or place extra terms in the objective functions.

The L^2 parameter norm penalty, also known as *weight decay*, adds a term to the cost function, for penalizing parameters $\boldsymbol{\theta}$, by "shrinking" them close to zero, thus preventing the network to model precisely the training data and, therefore generalizing better to new data samples:

$$\widetilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \alpha\Omega(\boldsymbol{\theta}), \tag{2.35}$$

where $\alpha \in [0, \infty)$ is a hyperparameter and $\Omega(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{w}\|_2^2$.

Dataset Augmentation is another regularization strategy that creates synthetic data, so that it can be added to training set, in order for the model to train with more data, thus creating a better generalization for the test dataset.

Early stopping is a regularization strategy that stops the training after a given stop criteria method, usually, after a number of "patience" epochs. In other words, when a given number of epochs does satisfy the stop criteria method ("patience" epochs) the algorithm only returns the parameters that did not satisfied the stop criteria method in the first place. For this decision to happen, usually, a part of the training set is removed in order to be used as a *validation set*. This validation set will be used to apply the stop criteria methods, mainly when performance starts to degrade in the validation set.

Dropout [40] is a powerful and recent method of regularization. It randomly selects and ignores nodes from the hidden layers each time it is loaded some mini batch during train, meaning that they will not contribute to the forward pass, neither updated in the backward pass. This causes networks to not overfit to training data. When testing, all neurons are used to make predictions, therefore Dropout works like a model averaging over all possible subnetworks that can be constructed by dropping out different subsets of units from the original network.

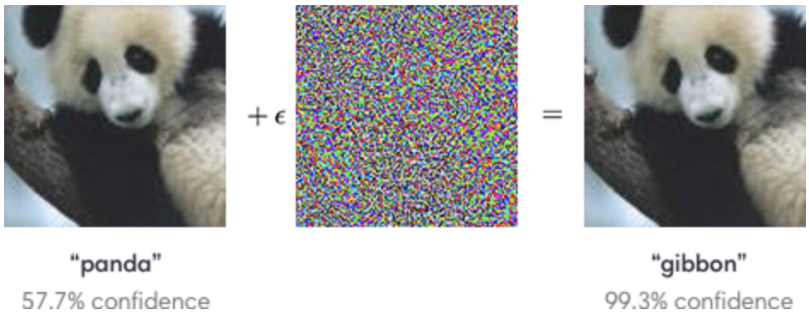


Figure 2.2: An adversarial input, overlaid on a typical image, can cause an image classifier to misclassify a panda as a gibbon. Source from [41].

DNNs began to reach human performance in many situations when evaluated on an independent and identically distributed (i.i.d.) test set. Therefore it is natural to wonder whether these systems have obtained some understanding of the underlying task. To check this, we could search for what kind of examples the model misclassifies. Szegedy et al. [42] concluded that DNNs are found to be rather fragile and easily fooled by *adversarial examples*. These adversarial examples are created by adding imperceptible noise to common examples, and thus are indistinguishable for humans. An example of this situation is depicted in figure 2.2.

Adversarial examples can be used as regularizers, via *adversarial training* – training the system on adversarially perturbed examples from the training set [41, 42].

2.2.2 Convolution neural networks

Convolutional neural networks (CNNs) [43] are a class of neural networks that use convolution in place of general matrix multiplication in at least one of the layers, and are mainly applied to data with a grid-like topology, e.g., images. CNNs were some of the first deep models to perform well by using back-propagation with gradient-based learning [44], and have been extremely successful in many practical applications [45, 46].

Convolution

The convolution operation is very straightforward. In a 2-D grid topology, a kernel matrix of weights, as depicted in figure 2.3, slides over the 2-D grid input, performing an element-wise multiplication in each current part of the input it is on, and then all values obtained are added together to create an output value.

The main motivations behind convolution are *parameter sharing*, *sparse interactions* and *equivariant representations*. Each of these ideas will be described, in more detail, below.

In a typical DNN, each weight matrix element is only used once when computing the output layer and never revisited. In a convolutional neural network, each weight of the kernel is used at every input position, except boundary pixels regarding design decisions. Therefore, the parameter sharing used by the convolution operation is more efficient in terms of memory requirements compared to the dense matrix multiplication. Mainly because k, the number of weights in the kernel, is usually smaller than the number of numbers in matrix $\mathbb{R}^{m*n}$ (where m is the number of inputs to the layer and n the number of outputs).

In typical fully connected DNNs, every output unit interacts with all input unit. However, CNNs have sparse connections, in the sense that only a subsection of pixels is used as input. For this reason, fewer parameters need to be saved in memory and that to compute the output, a small number of operations is applied. Matrix multiplication in dense layers run in $\mathcal{O}(n * m)$ for each example, while with convolution computations the performance is $\mathcal{O}(k * n)$, where k is smaller than m.

Parameter sharing in CNNs causes the layers to have a property called equivariance to translation. Let f be a convolution function and g a function that translates the input. The convolution function f is

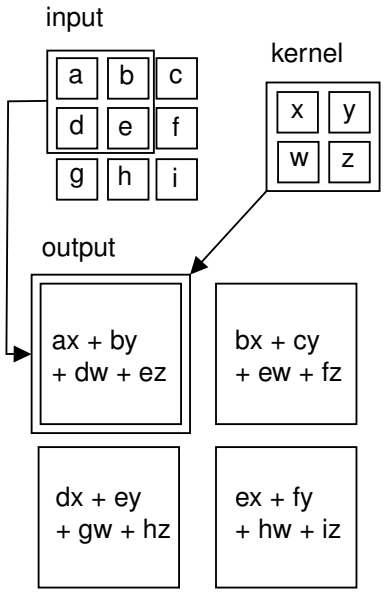


Figure 2.3: An example of 2-D convolution. The boxes with arrows indicate how the convolution is performed by applying the kernel to the input upper-left region, creating the output.

equivariant to g if

$$f(g(x)) = g(f(x)). \tag{2.36}$$

Despite this equivariance to translation, the convolution operation is not equivariant to rotations and scaling.

Pooling

A *pooling function* is commonly applied after a nonlinear function, e.g., ReLU, to further modify the output. This nonlinear function, commonly applied on top of a convolution operation, creates a down-sampling, invariant to small translations, of the input. This is useful when we only care about whether some feature is present, rather than exactly where it is. Some of the most common used pooling operations are *max pooling* [47] and *average of a rectangular neighborhood*. Max pooling reports the maximum output within a rectangular neighbourhood.

Before illustrating an example of a well-known CNN architecture, some definitions must be mentioned. Commonly, a kernel of dimensions $K \times K$ is applied to an input of size $I \times I \times C$, containing C channels, therefore, the volume of the kernel applied to the input image is of size $K \times K \times C$. The output produced will be of size $O \times O \times 1$, where O will be shown below how to compute it. It is important to note that C is usually omitted when mentioning kernel dimensions. There is also the *stride*, S, which denotes the number of pixels by which the window of the kernel moves after each convolution operation. *Zero-padding* denotes the process of adding P zeros to each side of the boundaries of the input. The most common modes are *valid padding* and *same padding*. Valid padding means that there is no padding. Same padding technique applies padding to the input (if needed) to fully cover the kernel and specified

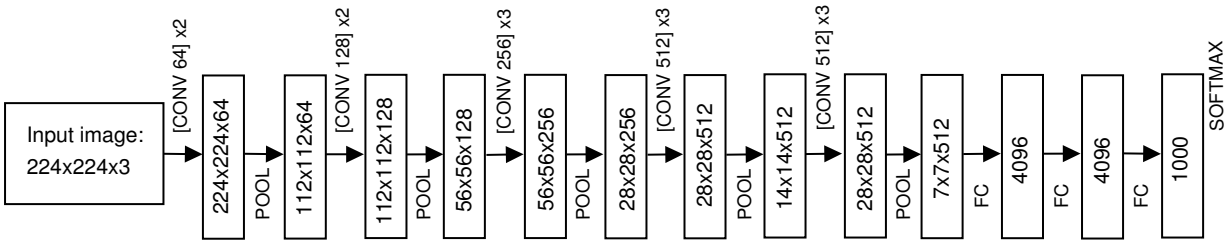


Figure 2.4: VGG-16 full architecture. All CONV layers represent a convolution, with kernel of size 3x3, stride 1 and same padding (height and width are the same), and a ReLU applied after the convolution. [CONV 64] means that there are 64 kernels that are going to be used, and, as a consequence, the depth of the next layer will be of that size. [CONV 64] x2 means that we have 2 consecutive layers of [CONV 64]. All POOL operations represent a max pooling function with a kernel of size 2x2 and stride 2. In the end we have 3 fully connected layers (FC), as mentioned in section 2.2.1, and a softmax applied in the last layer over the 1000 classes. It is called VGG-16 because there are 16 layers with weights (the CONV and FC layers).

stride. If stride is 1, the output size (width and height) will be the same as the input dimensions. Finally, the formula that computes the size of O is

$$O = \left\lfloor \frac{I + 2P - K}{S} \right\rfloor + 1, \tag{2.37}$$

where I is the input size, P is the padding size, K the kernel size and S the stride.

In figure 2.4 there is an example of a very well known CNN architecture named VGG-16 [48], with convolution and pooling operations.

2.2.3 Recurrent neural networks

Recurrent neural networks (RNNs) [49] are a class of neural networks that allow cycles, as opposite to feedforward networks. Thus RNNs become specialized for processing a sequence of values $\boldsymbol{x}^{(t)}$, where t is the time step, ranging from 1 to τ. These sequences are data points related in time, e.g., segments from audio, frames from video and words from text sentences.

Some of the most important and basic design patterns for recurrent neural networks are: *sequence to sequence*, where recurrent networks produce an output at each time step t, as illustrated in figure 2.5; and *sequence to one*, where recurrent networks only produce a single output after processing the entire sequence. Additional RNN designs, such as LSTMs and encoder-decoder models, relevant for our work, will be discussed in more detail at the end of this section.

As feedforward networks have an universal approximation theory, RNNs also can map any measurable sequence to sequence mapping to arbitrary accuracy [50]. Nevertheless, the main point is that the recurrent connections allow ”memory” of previous inputs to persist in the network internal state, thus influencing the final output.

The forward propagation equations for the standard RNN, depicted in figure 2.5 work as follow:

$$\boldsymbol{h}^{(t)} = \phi(\boldsymbol{b} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)}), \tag{2.38}$$

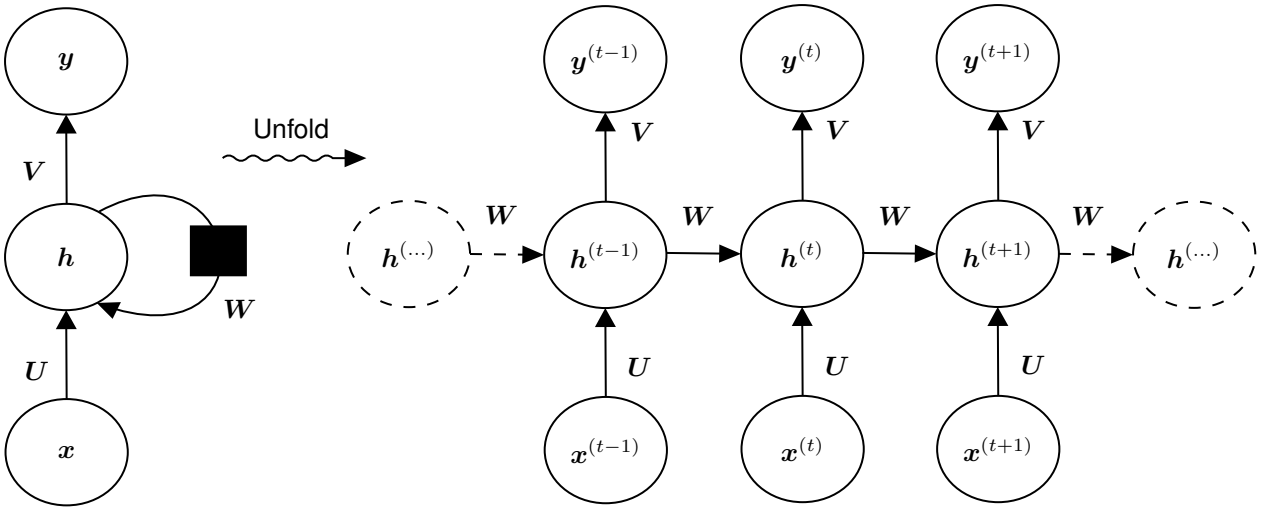


Figure 2.5: Unfolding of a recurrent neural network along the input sequence. Adapted from [9].

$$\hat{\mathbf{y}}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \tag{2.39}$$

where $\boldsymbol{b}$ and $\boldsymbol{c}$ are the biases, and $\boldsymbol{W}$, $\boldsymbol{V}$ and $\boldsymbol{U}$ are the weight matrices for the hidden to hidden, hidden to output and input to hidden connections, respectively. These equations are applied for each time step t, considering $\boldsymbol{h}^{(0)}$ as the initial state.

After the forward propagation, the gradient of the cost function J is computed and conducted from the right side to the left side of the unrolled network of figure 2.5 using an algorithm called *back-propagation through time* (BPTT) [9]. This algorithm is a generalization of back-propagation used for feedforward networks.

Long-short term memory

In practice, training an ordinary RNN, as the one in figure 2.5, often leads to the so called vanishing gradient problem [32]. One of the most effective sequence models that address this issue is named *gated RNNs*. Gated RNNs learn to decide what information is irrelevant and what information is important for the network, by clearing and updating the internal states, respectively. This is done using gate units.

Long short-term memory (LSTM) [51] recurrent network is a class of gated RNNs, that is capable of learning long term dependencies and has been found to be extremely successful on a large variety of tasks [10, 52, 53].

An LSTM network is the same as a standard RNN, except that it has a memory block in the place of the summation units of the hidden layer. The equations for the forward propagation are as follow:

$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{W}_f\boldsymbol{x}^{(t)} + \boldsymbol{U}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f), \tag{2.40}$$

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_i\boldsymbol{x}^{(t)} + \boldsymbol{U}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i), \tag{2.41}$$

$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{W}_o\boldsymbol{x}^{(t)} + \boldsymbol{U}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o), \tag{2.42}$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot tanh(\boldsymbol{W}_c \boldsymbol{x}^{(t)} + \boldsymbol{U}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_c), \tag{2.43}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot tanh(\boldsymbol{c}^{(t)}), \tag{2.44}$$

where $\boldsymbol{f}^{(t)}$ refers to the forget gate, $\boldsymbol{i}^{(t)}$ to the input gate, $\boldsymbol{o}^{(t)}$ to the output gate, $\boldsymbol{c}^{(t)}$ to the cell gate and $\boldsymbol{h}^{(t)}$ refers to the next hidden state. In plain terms, the forget gate decides what information is going to be forgotten from the cell state, the input gate combined with the tanh equation from 2.43 is going to decide which information is going to be stored in the cell. Equation 2.43 reflects the information that is going to be stored in the current state and equation 2.44 represents the information that is going to travel to the next states.

These gates allow LSTM memory cells to store and access information over long periods of time, therefore, mitigating the vanishing gradient problem, by keeping it flowing for long duration paths [54]. A simple solution that keeps away exploding and vanishing gradients is *gradient clipping*. The gradient is clipped with a reference value, before the parameter update, e.g., if the clipping value is 5 and the parameter gradient is -10, it will be replaced by -5. Gradient clipping, in other words, will force gradient descent to not go to a far region where the objective function is larger, therefore, not undoing much of the work it has been done to arrive at a solution.

Encoder-decoder architecture

Encoder-decoder architecture is an RNN trained to map an input sequence to an output sequence, in which both sequences can have distinct lengths. These kind of architectures are widely used in speech recognition and machine translation tasks, where the input and output, overall, are not of the same length, but might be correlated in some manner. First proposed in [55], the key concept of this architecture is very straightforward. Initially, a RNN encoder processes the input sequence, $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_{N_x})$, followed by the creation of the context $\boldsymbol{C}$, usually a fixed-size vector. Next, the RNN decoder generates the output sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, ..., \boldsymbol{y}_{N_y})$, based on $\boldsymbol{C}$. The two RNNs (encoder and decoder) are trained jointly to maximize $logP(\boldsymbol{y}_1, ..., \boldsymbol{y}_{N_y} | \boldsymbol{x}_1, ..., \boldsymbol{x}_{N_x})$.

One drawback of this design, observed in [56], is when context $\boldsymbol{C}$ has very small dimensions to adequately summarize a long input sequence, which reduces the performance of the encoder-decoder model. One solution proposed by [56] was to make $\boldsymbol{C}$ of variable length instead of a fixed size sequence and it was further suggested an attention mechanism, which is going to be more detailed in section 5.1.

Memory networks

Intelligence requires knowledge, and knowledge can be acquired via learning, which motivated deep neural networks development. However, DNNs struggle to memorize simple facts. In reality, we observe that SGD requires that one specific example goes as input to the model several times before it can be stored in the parameters of the neural network [9]. Graves et al. [57] conjectured that this happens because DNNs do not have a *working memory* system which explicitly holds and manipulates pieces of

information that are relevant for a particular task, similar to the human brain.

External memory networks were first introduced by Weston et al. [58], where a supervision signal was required to instruct how the memory cells would be used. To solve this dependence, neural Turing machines (NTMs) were introduced in [57]. NTMs can read and write arbitrary content to memory cells, without any supervision, by using a soft attention mechanism. Furthermore, they are fully differentiable, which makes it possible to train them in an end-to-end approach.

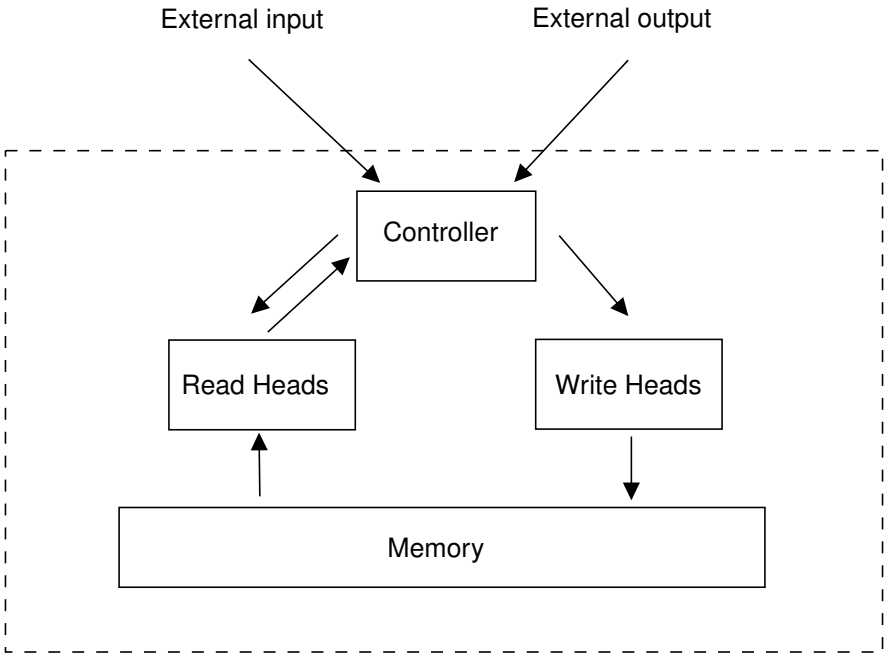


Figure 2.6: Neural Turing machine architecture. Adapted from [57].

The overall architecture of the NTM, which resembles the Turing machine invented in 1936 by Alan Turing, is depicted in figure 2.6, where the controller is usually an MLP or RNN which receives inputs, read vectors and emits outputs in response. Although the controller could be feedforward or recurrent, the overall system is always a recurrent network. The controller reads and writes from an external memory matrix via a set of parallel read and write heads. Next, the memory is a matrix $\boldsymbol{M} \in \mathbb{R}^{N \times W}$, where N is the number of memory locations (rows) and W is the vector size of each location (columns). In order to read at time t, a weighted average sum of all memory locations is performed, i.e.,

$$\boldsymbol{r}_t \stackrel{\text{def}}{=} \sum_i w_t(i) \boldsymbol{M}_t(i), \tag{2.45}$$

where $w_t(i)$ is the weight associated to row i, and $\boldsymbol{m}_t(i)$ is the memory vector from row i. Also, the sum of all weights adds up to one. For the write operation at time step t, two main steps are involved. The first step is an erase phase, i.e.,

$$\boldsymbol{M}_t(i)^{'} = \boldsymbol{M}_{t-1}(i)[\mathbf{1} - w_t(i)\boldsymbol{e}_t], \tag{2.46}$$

where $\mathbf{1}$ is a vector of ones and $\boldsymbol{e}_t \in \mathbf{R}^W$ is an erase vector. At last, the second step is an add phase:

$$\boldsymbol{M}_t(i) \stackrel{\text{def}}{=} \boldsymbol{M}_t(i)^{'} + w_t(i)\boldsymbol{a}_t, \tag{2.47}$$

where $\boldsymbol{a}_t \in \mathbf{R}^W$ is an add vector.

The weights mentioned above are computed by combining two addressing mechanisms: *content-based addressing* and *location-based addressing*. More loosely, content-based addressing is analogous to how people recall the full poem based on a few words, whereas location-based can be thought as "retrieving the full poem present in slot 9". These addressing mechanisms are identical to the attention mechanisms, briefly mentioned above when talking about the Encoder-decoder architectures, and discussed with more detail in section 5.1.

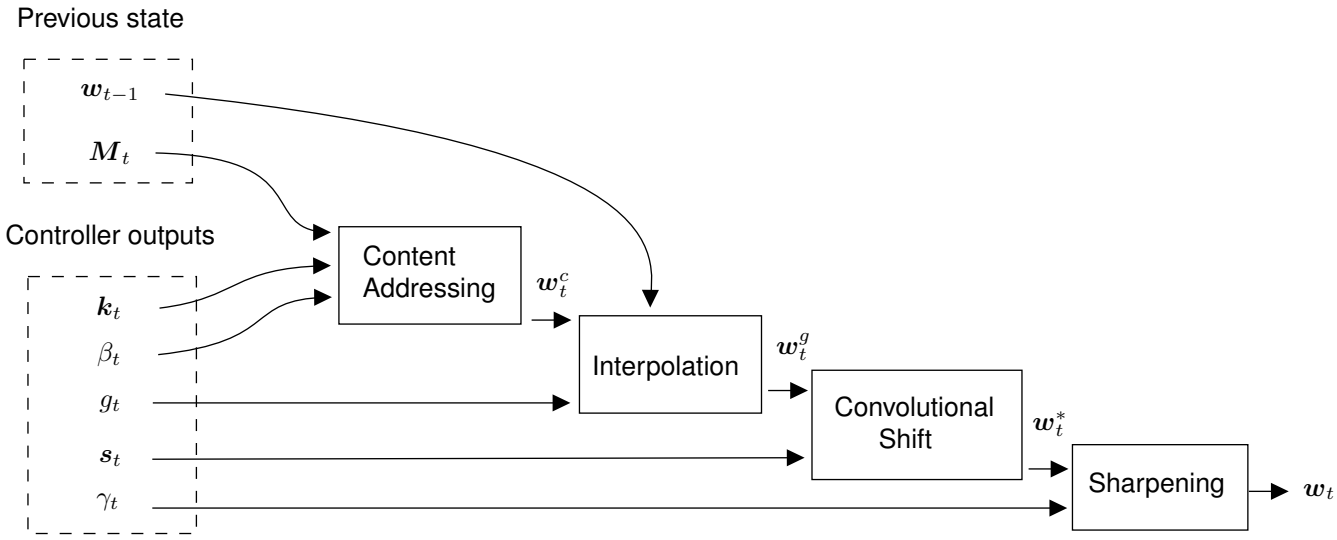


Figure 2.7: Flow diagram of the addressing mechanism in NTMs. Adapted from [57].

Figure 2.7 includes all intermediate steps to compute the final attention focus vector, i.e., $\boldsymbol{w}_t$. It is interesting also to note that the controller network outputs the necessary parameters to integrate different addressing mechanisms at different stages. First, it is measured the similarity between $\boldsymbol{k}_t$ and each entry of the memory, $\boldsymbol{M}_t(i)$, by using cosine similarity:

$$K[\boldsymbol{u}, \boldsymbol{v}] \stackrel{\text{def}}{=} \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\|_2 \|\boldsymbol{v}\|_2}. \tag{2.48}$$

By applying cosine similarity and softmax over the rows $\boldsymbol{M}_t(i)$, the weights for the content addressing are obtained as follows:

$$\boldsymbol{w}_t^c(i) \stackrel{\text{def}}{=} softmax(\beta_t K[\boldsymbol{k}_t, \boldsymbol{M}_t])_i, \tag{2.49}$$

where β_t, outputted by the controller as shown in figure 2.7, is a positive scalar parameter that determines how concentrated the content weight vector should be, i.e., for small values of β_t, the weight vector will be diffuse. However, for larger beta values, the weight vector will be concentrated on the most similar row in memory. The following steps are focused on the location-based addressing, and to implement it, we will need three more steps. The second stage creates $\boldsymbol{w}_t^g$ by interpolating $\boldsymbol{w}_t^c$ with the weight vector from last time step, $\boldsymbol{w}_{t-1}$, using $g_t \in (0, 1)$. This allows the system to learn when

to use or ignore content-based addressing. Next, in order for the focus of the weights to be shifted to different rows, the controller emits a shift weighting vector, s_t, that defines a normalised distribution over the allowed integer shifts. Each element in this vector gives the degree by which different integer shifts are performed. For example, if shifts of -1, 0, 1 are allowed a (0.1, 0.4, 0.5) shift-vector would denote a shift of 1 with strength 0.5, a shift of 0 (no-shift) with strength 0.4 and a shift of -1 with strength 0.1. The actual shift is performed with a circular convolution:

$$\boldsymbol{w}_t^*(i) \stackrel{\text{def}}{=} \sum_{j=0}^{N-1} \boldsymbol{w}_t^g(j)\boldsymbol{s}_t(i-j). \tag{2.50}$$

Next, if shifts of -1, 0 and 1 are given weights (0.1, 0.8, 0.1), the rotation will transform a weighting focused at a single point into one slightly blurred over three points. To solve this, a sharpening parameter $\gamma_t \geq 1$ is required:

$$\boldsymbol{w}_t(i) \stackrel{\text{def}}{=} softmax(\boldsymbol{w}_t^{*\gamma_t})_i. \tag{2.51}$$

Finally, we have a weight vector, $\boldsymbol{w}_t$, that determines where to read from and write to. NTMs inspired many other works based on memory external systems [59–61]. However, these systems tasks are usually simple actions, like copying and emulating simple n-gram models.

2.2.4 Practical methodology

To apply deep learning successfully, one requires more than just knowledge about the algorithms and the theory behind it. It is essential to know how to monitor and respond to feedback acquired from the experiments to improve the machine learning model. Blindly guessing what do next is not enough to determine the right path in the practice of deep learning. This section will describe briefly the methodology used for this work when using deep learning. All techniques come from Andrew Ng book [4].

The main design process consists on (1) first determining the goals, what single-number metric to be used, associated with a target value for this specific error metric, and what are going to be the validation and test sets. For the target value, references like human-level error or Bayes error (minimum error rate possible to achieve) are commonly used. The second main step is to (2) establish a simple working system as soon as possible. If possible, it is an excellent approach to use an architecture that is already successfully used in the same kind of problem. After building the first model, the next step is to (3) apply bias/variance (described below) and *error analysis*, i.e., examine the examples that were misclassified in the validation set, so that conclusions can be made about what were the underlying causes. After using bias/variance and error analysis, (4) it is essential to do incremental changes such as adjusting hyperparameters, gathering and adjusting data and, the most demanding one, creating a new algorithm or architecture. From 1 to 4, all these steps can be redone all over again if, for example, in step 4, we conjecture that the metric being used is not the most appropriate.

It is important to note that the term bias and variance used here, for simplicity and as mentioned in [4], do not follow the formal definition used in statistics. The error rate on the training set is used as

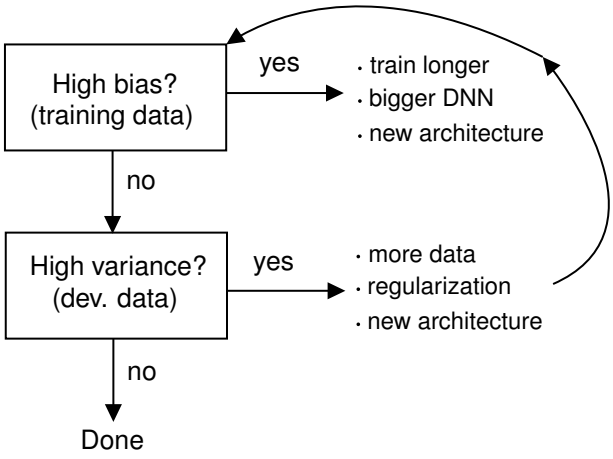


Figure 2.8: Bias/variance machine learning practice. From `https://www.deeplearning.ai`, by Andrew Ng.

the algorithm bias, and the variance is how much worse the system performs when comparing training with validation or test set. For example, suppose that a model recognizing speech has 16% error on the validation set and 15% on the training set. We can conclude that the variance, i.e., 1%, is small when compared to the bias. The main bias/variance workflow used in this project is depicted in figure 2.8.

2.3 Automatic speech recognition

The first genuinely complete speech recognition system, named ”Audrey”, was created in Bell Labs, 1952. It could only recognize digits. In 1962, IBM demonstrated ”Shoebox” machine [3] in Seattle world fair, that could only understand 16 words spoken in English. From 1971 to 1976, the Defense Advanced Research Projects Agency (DARPA) created funds for speech recognition research to end up with a machine capable of understanding a minimum of 1000 words. As a result, Carnegie Mellon University (CMU) built ”Harpy”, a system that could understand 1011 words. Since the early 1980s, the speech recognition community started to use HMMs to predict the most probable symbols given the last one. Ever since the community of automatic speech recognition (ASR) has been growing and improving.

In this section, the modern goal of ASR will be defined, followed by a brief explanation of some of the main challenges that arise in the field, how acoustic features are commonly extracted. Finally, an overview at the two most common ASR approaches, i.e., *HMM-based* and *end-to-end*, will proceed. Details about these two speech recognition paradigms will be presented later in chapters 4 and 5, respectively.

2.3.1 Automatic speech recognition goal

The goal of speech recognition is to map an acoustic signal, containing utterances from a spoken natural language, into the correct sequence of words, without being necessary to figure out the meaning of what

[3] https://www.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html

was said. Let $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(T)})$ be a sequence including acoustic input vectors, usually formed by splitting the audio into 25ms frames, and $\boldsymbol{y} = (y^{(1)}, y^{(2)}, ..., y^{(T)})$ denote the target output sequence, commonly a sequence of words or letters. The process of speech-to-text transcription, denoted ASR, consists of creating a function f^* that calculates the most probable sequence of words, sub-words or letters $\boldsymbol{y}$ given $\boldsymbol{X}$:

$$f^*(\boldsymbol{X}) = arg\max_{y} P(\boldsymbol{y}|\boldsymbol{X}) \tag{2.52}$$

$$= arg\max_{y} log\ p(\boldsymbol{X}|\boldsymbol{y}) + log\ P(\boldsymbol{y}) \tag{2.53}$$

Equation 2.53, derived from applying Bayes rule and applying a log domain, divides the problem into two: *language modeling* (LM) and *acoustic modeling* (AM). Language modeling is concerned with computing the prior probability $P(\boldsymbol{y})$ whereas acoustic modeling estimates the likelihood $p(\boldsymbol{X}|\boldsymbol{y})$.

The proper evaluation measure for the task of transforming speech signal into words is denoted as *word error rate* (WER). The sequence of output words is aligned using a dynamic programming alignment algorithm with the target sequence and, once that is performed, the accuracy of the ASR can be measured as a string edit distance between the real output and the desired target output (containing N words) by counting the number of *S* substitutions, *D* deletions and *I* insertions. WER can be defined as:

$$WER = 100 \times \frac{S + D + I}{N}\% \tag{2.54}$$

and the corresponding accuracy as:

$$Accuracy = (100 - WER)\% \tag{2.55}$$

Finally, for the character error rate (CER) and phone error rate (PER), the equations are similar to the WER equation. The main difference is that the evaluation corresponding to CER is done at the character level, and the evaluation corresponding to the PER is done at the phoneme level. Therefore the N in equation 2.54 for CER is the number of characters in the desired target output, and for the PER it is the number of phonemes present in the desired target output.

2.3.2 Automatic speech recognition challenges

There are several causes of variability that make the task of ASR arduous [62]. Firstly, we have the *size of the vocabulary*, which mainly depends on the language being used, e.g., agglutinative languages - for example Hungarian, Finnish, Japanese and Korean - have larger vocabularies than non-agglutinative languages, like English. Another source of variability is the speaker, i.e., all speakers have differences that arise from the combination of vocal tract, age and accent. One way to deal with this vast variability, so that the system can work with more speakers, is to create a *speaker-independent* instead of

a *speaker-dependent* system. The *acoustic environment* can also have a big impact on the accuracy of the ASR model when we are dealing with the presence of noise, competing speakers, echo and reverberance. For instance, if an ASR system is trained under close to perfect acoustic environment conditions and, for example, someone seated on a couch wants to use it when the TV is on, the ASR system will behave badly. Finally, the *speaking style* is the main last source of variability that makes ASR task more challenging. If we want an ASR system that has good accuracy performance with both *continuous* and *spontaneous* speech, as opposite to an *isolated word* and *planned* speech, respectively, the path to achieve it will be harder. In particular, while speech from reading newspapers and books can be recognized with high accuracy, for spontaneous speech, recognition accuracy drastically decreases because of the higher variability in it caused by disfluencies, like filled pauses, repetitions, false starts and ”repaired” utterances. This higher variability translates into the need of having a much larger corpus or a more optimized or new learning model that encompass all variations.

2.3.3 Acoustic features

The speech waveform is not directly used by speech recognition systems since raw audio data is noisy and meaningless. Instead, techniques of signal processing are used to derive the acoustic features that are going to be modeled by an HMM-based or by an end-to-end model. These acoustic features are a compact representation of the wave form that only contains essential signal information needed to discriminate between words, and they are commonly computed every 10ms, using a Hamming window, i.e., an overlapping analysis window of around 25ms [62]. Despite of a large variety of representations used in speech recognition [63], *mel frequency cepstral coefficients* (MFCCs) are the most widely known and used. Introduced by Davis and Mermelstein in the 1980's [64], these are originated by applying a Fourier Transformation to each frame of the original speech signal, which changes the signal from a time domain to a frequency domain. A mel scale is applied, using triangular overlapping windows, that approximates the response of the human ear. After the mel scale, it is followed a log domain, since we do not hear loudness on a linear scale, and finally, a discrete cosine transformation (DCT) is performed [65]. At the end we get a set of vector features, normally 12 values for each frame, named as mel frequency cepstral coefficients. Besides these set of coefficients, something that improves considerably the recognition accuracy is the addition of the first and second temporal derivatives coefficients (also known as delta and delta-deltas), which extract temporal information between phoneme transitions not present in spectral features [65] (this compensates the assumption of conditional independence made by the HMMs, stated at 2.1). The final result after the mentioned augmentation of delta and delta-deltas is a feature vector of size around 40. The other common used representation is *filter banks*, where the main difference from MFCCs is that the last transformation applied is the mel scale.

It is known, as mentioned in section 2.3.2, that speech recognition accuracy decreases in noisy environments, so it is a must for speech recognition systems to be robust with respect to their acoustic environments. For example, if an acoustic model is trained using a set of features from a clean acoustic environment, a set of test features containing utterances from an acoustic noisy environment would

mismatch with the already trained model, therefore reducing its recognition accuracy. To overcome this problem, partially, *cepstral mean and variance normalization* (CMVN), [66] is commonly used. It is important to mention that if these methods are not correctly used, the ASR performance may decrease.

2.3.4 Two main paradigms in speech recognition

There are two main approaches for the task of ASR: *HMM-based* and *end-to-end* models. HMM-based models, discussed with more detail in chapter 4, were state-of-the-art for more than twenty years [9]. They consist of an *acoustic model*, a *language model*, a *pronunciation model* and a *decoder*. The acoustic model is used to model the mapping between speech input and feature sequence (usually phoneme sequences). The pronunciation model, built typically by specialized human linguistics, maps phonemes to graphemes, and the language model maps the character sequence to the final transcription. The construction and implementation of these modules are complex and time-consuming. Also, they are all optimized with different objectives. More importantly, these sub-optimal optimizations do not optimize the real goal of the ASR task, i.e., usually minimizing a distance metric error for the labels (usually words, characters or phonemes, e.g., WER). Since 1980, HMM-GMMs systems were widely used and extensively researched. Consequently, they started to approach their ceiling, especially when dealing with daily conversations, such as telephone calls and conferences. After 2012, state-of-the-art improved significantly in ASR by replacing the GMMs from the HMM systems with DNNs [8]. Beyond the sub-optimal training problem, the conditional independence assumptions taken into account (within the HMM and language model) to simplify the model training and construction do not match the real speech situation.

Due to the mentioned HMM-based problems, more and more work started to exist for end-to-end systems, discussed with more detail in chapter 5. Compared to the HMM-based system, the end-to-end system replaces multiple modules with a DNN that directly maps input audio sequence to a sequence of characters, words or sub-words. The above difference gives the following characteristics to the end-to-end system:

- Jointly training the multiple modules with a single network architecture enables the end-to-end system to use a function that seeks globally optimal results [10].
- It directly maps the acoustic features to the output text and does not require further processing to achieve the final transcription, as opposite to HMM-based models [67].

For end-to-end, there are two main approaches: *CTC-based* and *attention-based* models, which will be briefly explained below. The closest and first proposed approach to solve end-to-end in ASR was proposed by Alex Graves et al., in 2006 [14], with the creation of the CTC loss function which solves hard alignment by computing its loss with the auxiliary of a *blank* label. In other words, CTC loss first enumerates all possible hard alignments (denoted by the name path) and then it achieves soft alignment by aggregating these hard alignments. Also, CTC directly outputs the target transcriptions. CTC main problem is that it cannot model the language, because it considers each label in the output sequence

to be independent of each other. This is the reason why CTC-based models are not fully end-to-end. To solve the independence assumption of CTC-based models, the RNN-T model, which was proposed in [15], enumerates all hard alignments and then aggregates them for soft alignment. However, as opposite to CTC, it does not make independent assumptions about labels when enumerating the hard alignments.

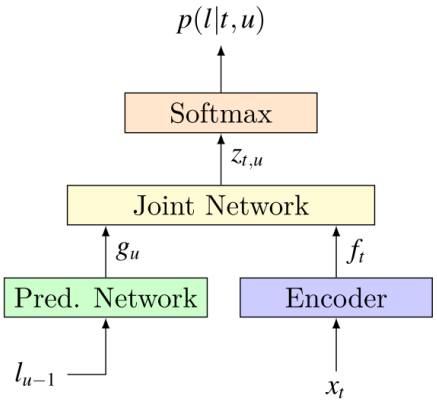


Figure 2.9: Recurrent neural network transducer (RNN-T) architecture. Source from [68].

The RNN-T network architecture, depicted in figure 2.9, consists of three subnetworks. The transcription network is an encoder that maps the acoustic features into a speech embedding space, and the prediction network is an RNN that models the interdependencies within the output label sequence. The third subnetwork is a joint network which creates the alignment between the input and output sequence. RNN-T loss is very similar to the CTC loss, in the way that their goals pass by solving hard alignment in speech recognition, by using dynamic programming. Despite the advantages over CTC, i.e., no independence assumptions exist anymore for modelling language, the RNN-T model is harder to train, thus using pre-training to improve the model performance is needed [69]. Another disadvantage in all CTC-based and especially in RNN-T models is that, since they first enumerate all hard alignments and then aggregate them, there could be many illogical paths.

Attention-based models are the other paradigm of end-to-end systems in ASR. As opposed to CTC-based and RNN-T, these systems do not enumerate all hard alignments. Instead, they do soft alignment between input and output, directly, by using an attention mechanism. The attention-based systems can be divided into an encoder that processes all input feature sequences. An attention layer that works as an aligner and the decoder that models the language. One of the main problems that arise with the combination of the encoder and attention modules is latency. Since the attention module needs to wait for the encoder to produce the full encoded sequence, in order to start working, there is an increase in the delay from the model. Conversely, the CTC-based and RNN-T models do not need to wait for the full encoded sequence produced by the encoder. Two drawbacks that come from this increased delay are: on the one hand, if the encoded sequence is significant, the attention module will take more time to do its computations, therefore increasing more the delay; on the other hand, since the speech is much larger than the ground truth transcription, much redundant information will be given to the attention module.

The solution for this problem and the monotonic problem introduction will be more detailed in chapter 5. Overall the attention-based models have better accuracy than the CTC-based or RNN-T models but have the most delay.

Chapter 3

Corpora

In this chapter, we provide a full description of the corpora used in all experiments for this thesis, reported in sections 4.2 and 5.2. Initially, for gathering data, unsuccessfully extensive research was performed on the web to find publicly available European Portuguese transcribed data resources. It was shortly discovered that most common public speech repositories, which were few, e.g., LibriVox, practically do not contain any European Portuguese content, as opposite to Brazilian and English languages. Consequently, it was decided to use some of the data sets owned by INESC-ID for automatic speech recognition purposes. This chapter starts with a description of the ALERT corpus, followed by a description of BD-PÚBLICO, SPEECHDAT and TRIBUS datasets.

3.1 ALERT

Collected from November 2000 through January 2001, ALERT [70] is a Portuguese broadcast news (BN) corpus that contains spontaneous speech. It was created with the cooperation of RTP, a public service broadcasting organisation from Portugal. This corpus, downsampled to 16kHz, is already divided into a training, validation and test set. It is also composed of two primary types of files: an audio stream file (.wav) and an associated transcription file (.trs). The training data contains around 60 hours of speech with 1366 speakers and 47552 utterances. The validation set contains 8 hours of data with 260 speakers and 6222 utterances. Finally, the test set with 6 hours has 175 speakers and 4701 utterances. All this information is depicted in table 3.1.

There is a 4-gram trained language model associated with this corpus, and a pronunciation model, with 115503 different pronunciations in SAMPA format [1], created both in AUDIMUS.media [70]. The language model was designed by using linear interpolation of three distinct LMS. The first is a backoff 4-gram LM, trained on a word corpus of newspapers texts containing 700M words. This out-of-domain corpus was collected from the web. The second LM is a backoff 3-gram LM trained on an in-domain corpus of broadcast news transcripts, with around 531k words. Finally, the third model is a backoff 4-gram LM, estimated from the European Portuguese web newspapers, collected the week before creating

[1] www.phon.ucl.ac.uk/home/sampa

the interpolated LM. The final interpolated LM is a 4-gram LM with Kneser-Ney modified smoothing, 100k 1-gram, 7.5M 2-gram, 14M 3-gram and 7.9M 4-gram.

	#utters	#speakers	#hours
training set	47552	1366	60
validation set	6222	260	8
test set	4701	175	6

Table 3.1: ALERT experimental speech corpus.

3.2 BD-PÚBLICO

BD-PÚBLICO [71], similar to Wall Street Journal corpus (WSJ0), consists of read speech from the Portuguese newspaper Público. Created in 1997, it contains 120 different speakers, where ages range from 19 to 28 years old. Thus it is a speaker-independent continuous speech recognition system. The training, validation and test sets are already divided into distinct subsets. The training set contains around 23 hours with 100 speakers and, the validation and test set contains 2 hours and 10 speakers each. The sampling rate, 16kHz, is the same as ALERT, mentioned above. BD-PÚBLICO also comes associated with a backoff 3-gram closed language model, trained using the CMU-Cambridge statistical language modelling (SLM) toolkit, and a SAMPA pronunciation model with 32144 different pronunciations. The language model being closed means that OOVs are eliminated from training data and are forbidden in test data.

	#utters	#speakers	#hours
training set	8389	100	23
validation set	584	10	2
test set	592	10	2

Table 3.2: BD-PÚBLICO experimental speech corpus.

3.3 SPEECHDAT

SPEECHDAT [72] corpus is a collection of speech read from telephone calls, collected by Portugal Telecom, a Portuguese telecommunications operator named Altice Portugal. Later, this collected data was designed and post-processed by INESC-ID. SPEECHDAT was recorded in two main phases: SPEECHDAT 0, with approximately 1000 speakers involved and the second was SPEECHDAT 1, with approximately 4000 speakers involved. Each telephone call included in the database contains 33 red items and 7 spontaneous answers, where some contain demographic information. Only the 9 phonetically rich sentences, from the set of 33 items, were used to create SPEECHDAT corpus for the ASR task.

As opposite to ALERT and BD-PÚBLICO, SPEECHDAT experimental setup had to be created. When working with SPEECHDAT we noticed that SPEECHDAT 1 contains only 3622 unique utterances from

the total number of 36243 and SPEECHDAT 0 has only 904 unique utterances compared to the total of 9000 utterances, also mentioned in [73]. Furthermore, SPEECHDAT 0 and SPEECHDAT 1 are two disjoint sets. Considering that, SPEECHDAT 1 was transformed into a training set and SPEECHDAT 0 into validation and test sets. For the validation and test sets, the data was divided according to the number of speakers, i.e., half of the speakers to the validation set and the other half to the test set. This data splitting process was made such that the number of female and male speakers were almost the same for each set. Since there are speakers where gender is not labelled in the original data set, the last set, i.e., test set, being created got all unlabelled speakers. As depicted in table 3.3, the training set contains around 63 hours, the validation set contains 9 hours, and the test set has 9 hours. Also, random seeds were used to create and recreate the validation and test sets with the same data. The sampling rate used for SPEECHDAT is 8kHz, as opposed to ALERT and BD-PÚBLICO, typical for telephone speech data.

For creating the pronunciation model, which contains all unique words that appear in the training set of SPEECHDAT, we collected a lexicon of 12115 words, using SAMPA for phonetic annotation, obtained from publicly available resources. Finally, we first estimated a back-off 3-gram model with Kneser-Ney smoothing combined with Good-Turing smoothing to create the language model. Using the validation set, the perplexity for this LM is 1182.28. Since this LM is a poor model to use for HMM-based ASR in SPEECHDAT, we interpolated this 3-gram LM model with BD-PÚBLICO LM. Using all combinations of weights, i.e., 0.1 to 0.9, we estimated the best language model obtaining a perplexity of 230 on the validation set. The best combination of weights, used for the linear interpolation, were 0.2 to the LM of SPEECHDAT and 0.8 to the LM of BD-PÚBLICO. It is relevant to note that, to train the linear interpolated LM, the pronunciation model of SPEECHDAT was combined with BD-PÚBLICO pronunciation model, giving a total of 33470 different pronunciations. This BD-PÚBLICO pronunciation model is different from the original mentioned above and is described with more detail in TRIBUS section 3.4.

	#utters	#speakers	#hours
training set	36243	4027	63
validation set	4497	500	9
test set	4503	501	9

Table 3.3: SPEECHDAT experimental speech corpus.

3.4 TRIBUS

TRIBUS training set, with 92131 utterances, was created by concatenating all training sets from SPEECHDAT, ALERT and BD-PÚBLICO, using a Kaldi [74] script named ”combine_data.sh”. For the validation and test sets, we used the same sets from SPEECHDAT, ALERT and BD-PÚBLICO. Hence, TRIBUS contains three validation and three test sets. The number of utterances, speakers and hours for the mentioned sets is depicted in table 3.4. It is relevant to note that for decoding, the language model used for each set is the one that corresponds to the respective corpus.

Next, in order to have one pronunciation model for TRIBUS, we took all different words from ALERT original pronunciation model and created a new one using the online word to phoneme converter, the same as in SPEECHDAT 3.3. Following, SPEECHDAT and ALERT pronunciation models were grouped to create a new one with around 99583 different pronunciations. Eventually, the pronunciation model was used to train a 5-gram grapheme to phoneme (G2P) model using sequitur G2P toolkit [2]. After training it, the model was used to make inference in all BD-PÚBLICO words. At last, all new pronunciation dictionaries, except for SPEECHDAT, were grouped, giving a total of 108358 unique pronunciations. It is important to note that, instead of using the online source to create the pronunciations for BD-PÚBLICO, the G2P model was created to have a system to generate every sequence of phonemes for all European Portuguese words in a fast and reliable way. Finally, to train and evaluate ASR systems with TRIBUS, ALERT and BD-PÚBLICO datasets were downsampled to 8 kHz, so that they would match the sampling frequency of the telephone data.

	#utters	#speakers	#hours
training set	92184	5493	146
ALERT validation set	6222	260	8
SPEECHDAT validation set	4497	500	9
BD-PÚBLICO validation set	584	10	2
ALERT test set	4701	175	6
SPEECHDAT test set	4503	501	9
BD-PÚBLICO test set	592	10	2

Table 3.4: TRIBUS experimental speech corpus.

[2] https://github.com/sequitur-g2p/sequitur-g2p

Chapter 4

Hidden Markov based models

This chapter main goal is to describe the creation of state-of-the-art HMM-based systems, respectively, for BD-PÚBLICO, SPEECHDAT, ALERT and TRIBUS, mentioned in chapter 3. For this purpose, all systems mentioned above will be trained using an HMM-GMM and an HMM-DNN model to compare both approaches. This chapter starts with a brief overview of some key concepts behind HMM-based models and how they evolved, in 4.1. Section 4.2 and 4.3 detail the experimental setup and the results obtained, respectively. Finally, in 4.4 we discuss some conclusions regarding HMM-GMM and HMM-DNN systems.

4.1 Related work

HMMs combined with Gaussian Mixture Models (GMMs) were the quintessential systems that dominated speech recognition state-of-the-art, from 1980 until 2012. While HMMs naturally model the temporal sequence of phonemes, GMMs dictate how each hidden state of the HMM fits into the acoustic feature vectors mentioned in section 2.3.3 [75].

In HMM-based systems, the phoneme is the elemental unit of sound used by the acoustic model, e.g., the Portuguese word ”galho” is composed by four phones /g/ /a/ /lh/ /o/. These traditional ASR systems, are comprised of an acoustic model, language model, pronunciation model and a decoder as depicted in figure 4.1. Each model is independent of each other and all must be trained solely. It is also worth noting that these individual training can be very complex and time consuming for each model, respectively.

As mentioned in beginning of section 2.3.1, the acoustic model is used to model the mapping between speech input and acoustic features and the language model to compute the probability of a sentence, or sequence of words, of a given language, which in most cases is a n-gram model, that computes the most probable word that might follow after the last n-1 words [76]. Next, the pronunciation model, which requires a lot of human expertise, maps words from the vocabulary to their correspondent phonemes. It is important to note that there are words which have the same writing but different pronunciations (e.g., the word ”colher” in Portuguese can be used to talk about the spoon used in the

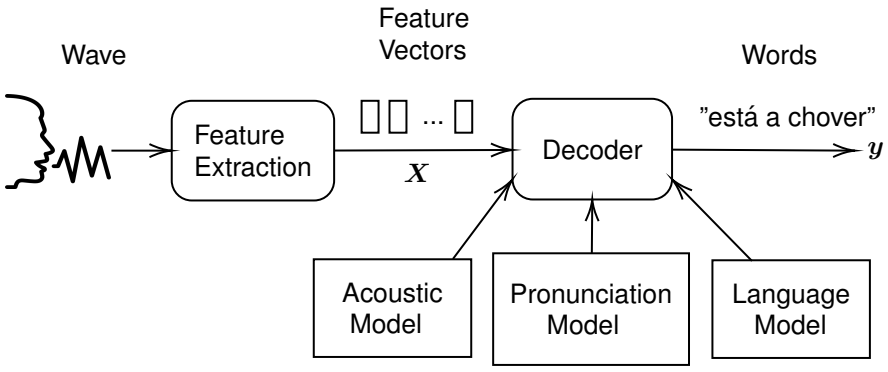


Figure 4.1: HMM-based architecture.

kitchen or to talk about harvest, with different pronunciations) and that the HMM-based model will only recognize existing words present in the language vocabulary, that is specified when a LM is trained. Lastly, the decoder searches for the most likely sequence of words or characters given the acoustic model, the language model and the pronunciation model, mentioned above. In most of the cases we are in the presence of a large vocabulary continuous speech recognition (LVCSR), which makes the search harder, because of the huge searching space, that might be much bigger when dealing with long-span language models, like trigrams. Viterbi algorithm, mentioned in section 2.1 works well for small vocabularies, but finding a solution for LVCSR using only Viterbi is unfeasible. Optimized decoders use pruning to remove unlikely hypotheses, thereby keeping the search more tractable. Some examples of these optimized decoders are based on *beam search*, *weighted finite state transducers* (WFSTs) theory or in *stack decoding*, which is essentially an A*-search [62].

It is also pertinent to point out that, in the presence of LVCSR, powered by enhanced computing ability and big data, HMM-based systems work with *context-dependent* phone models, often called triphones, i.e., that have both a left and a right context. If the HMM model was built with only phone models, also known as *monophones*, all output speech words would be formed by concatenating a specific sequence of phone models together. The issue that emerges when using this system is that the mapping from phonemes to words is not unique. For instance, the word "colher" mentioned above in this section has a different sequence of phonemes for the two different pronunciations in European Portuguese (spoon or harvest), and also, the sentence "It's hard to recognize speech" can be transcribed as "It's hard to wreck a nice beach" if using monophone models. This means that pronunciation contexts are different depending on the preceding and following phonemes of the word, hence the importance to have a model that can capture the large variability of speech, mainly when working with LVCSR, i.e can solve problems like coarticulation (changes in speech articulation as a result of the neighboring speech). For triphones, if there are N possible phones in the respective language, there are N^3 potential triphones. Since there will probably exist many triphones that are not observed in the training data, a sharing of parameters between different triphones must be performed (illustrated in figure 4.2). Usually this aggregation that creates tied-state triphone models is done through classification and regression trees (CARTs) [62].

Despite of the prominent dominance of HMM-GMMs systems from 1980 to around 2012, neural networks were already used in some recognition systems, with strong comparable results for HMM-

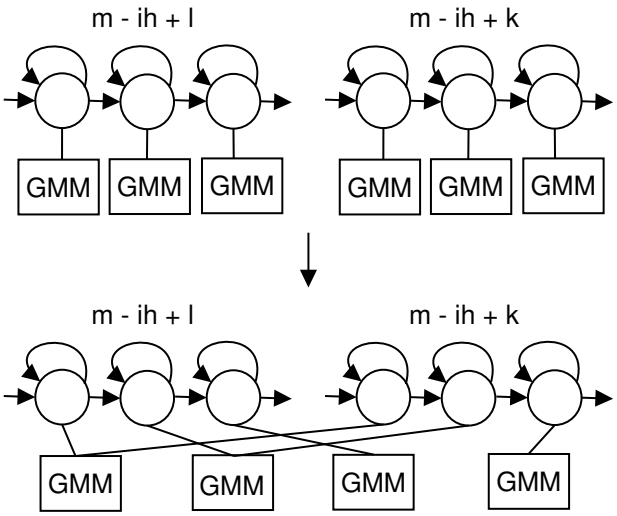


Figure 4.2: Formation of tied-state triphone models.

GMMs models [77], using TIMIT [21] corpus as a comparison base term. From that moment on, TIMIT has been a standard for phoneme recognition. Despite all of this, the switch in the industry to neural networks was not justified because of all investment expenditure done for HMM-GMMs systems [9]. However, as a result of this extensively research in HMM-GMMs systems, the performance was also approaching its ceiling, more specifically with spontaneous speech.

In the course of time, advances in computer hardware were made, which enabled powerful methods for training deep neural networks, that enclose many layers of non-linear hidden units. As a result, GMMs started being replaced by DNNs, because the latter have the potential to learn much better models of data that lie on or near a nonlinear manifold compared with GMMs, i.e., they can statistically generalize better [8]. The early approach adopted in 2009 used *restricted Boltzmann machines* (harmonium as the original name for these deep probabilistic models [78]) to model the input data in an unsupervised way. After the pretraining phase, they are used to initialize the DNN layers parameters, an important optimization technique for non shallow networks, and then the whole network is discriminatively fine-tuned to predict the target HMM states for a given fixed-size input frame. The phoneme error rate, on TIMIT, went downwards to 23.0 percent with this new approach [79]. Later, in 2012, Hinton et al. [8] portrayed all progresses achieved by using deep learning in the recognition field. This overall switch, translated into an improvement of around 30 percent on the performance of ASR systems. From this moment on, the speech recognition community headed for the use of DNNs in acoustic modelling. Currently, time-delay neural networks (TDNNs) [80], similar to convolution operations, discussed in 2.2.2, applied over temporal data, with stride 1, no pooling and no padding, is one of the most commonly used architectures that replace GMMs. In 2015, a new architecture was proposed in [81], where the main difference from original TDNNs is that the outputs of the activation from previously hidden layer are spliced as input to the current layer. Therefore, the current layer operates at a much broader context, compared with the previous layers. As we can see from figure 4.3, from layer 3, if we have $\{-3, +3\}$, it means we splice together the input at the current frame minus 3 and the current frame plus 3. A more recent version of TDNN which gives better performance results is TDNN-F [82], whose matrix weights

from a layer are compressed using singular value decomposition (SVD), i.e., creating a bottleneck layer.

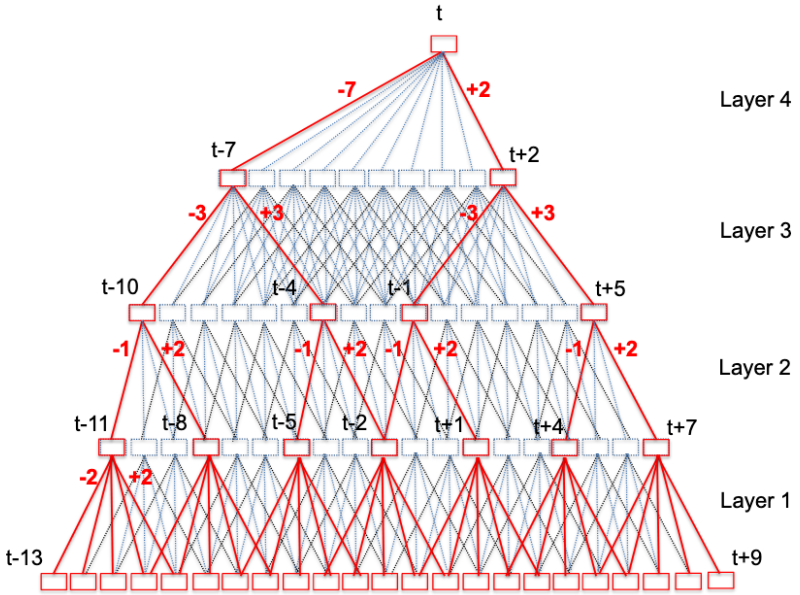


Figure 4.3: TDNN with subsampling. Source from [81].

Despite these accomplishments, the role played by DNNs was still limited, since it only models the posterior probability of the HMM hidden states. HMMs also have weaknesses working as generative models, mainly the lack of correlation between acoustic features and the temporal independence of speech between HMM states. It is also relevant to note that, to train a state-of-the-art HMM-DNN system, it is first necessary to train an HMM-GMM system to align the corresponding frames and HMM triphone states. Thus, there is a huge strive by the community to conceive new ASR architectures using only deep learning frameworks. A modern approach named end-to-end is discussed with more detail in chapter 5.

4.2 Experimental setup

To compare both approaches of HMM-GMM and HMM-DNN models, experiments were conducted using Kaldi [74] (a free, open-source toolkit for speech recognition research) for BD-PÚBLICO, SPEECHDAT, ALERT and TRIBUS, respectively. The four European Portuguese corpora last-mentioned are described in chapter 3. It is also important to note that all recipes used to train the HMM-based models, for each corpus, come from the same Kaldi baseline: Wall Street Journal (WSJ) recipe, available at [1].

The main experimental procedure consists on first training all HMM-GMM models in each respective corpus mentioned above, and then, using the alignments created for each corpus, train an HMM-DNN model, e.g., we first train an HMM-GMM model with ALERT corpus and then use the alignments to train an HMM-DNN system on the same dataset, and so on for all remaining datasets. The workflow for each HMM-based model will be more detailed in section 4.2.2 and section 4.2.3. Before explaining the training flow, we will give a description of relevant Kaldi data preparation for each corpus.

[1] https://github.com/kaldi-asr/kaldi/tree/master/egs/wsj

4.2.1 Kaldi initial setup

Before training, it is required to create some Kaldi files at the beginning of data preparation for the recipes to work, beyond the main ones: text (mapping from utterance id to transcription), wav.scp (mapping from utterance id to the filename or a command that extracts a wav format file) and utt2spk (mapping utterance id to speaker id) files. First, it is required to create one file named lexicon.txt which contains all mappings from words or non-lexical words (e.g., _laugh_ and _ehm_hmm_, where the former means laugh and the latter is an interjection) to the corresponding sequence of phonemes. It is relevant to note that each non-lexical (special) word created for all corpora mentioned in chapter 3, will have one specific matching phoneme. Before going into more detail about the other files, all non-lexical words and respective phones (between brackets) created for all corpora, i.e., BD-PÚBLICO, SPEECHDAT, ALERT and finally TRIBUS, when required, are:

- _ehm_hmm_ [ehm], which maps to all interjection sounds.
- _inahle_ [inh], which maps to all inhaling sounds.
- _laugh_ [lau], which maps to all laugh sounds.
- _nsnoise_ [nsn], which maps to all non-spoken noise sounds.
- _spnoise_ [spn], which maps to all spoken noise sounds.

It is also essential to notice that there is another special symbol, which represents silence, named _sil_. This symbol represents already a phone by itself. Following, it is required to create a file named nonsilence_phones.txt, with a list of all phones corresponding only to the words and not special words, and a file named silence_phones.txt, which contains all phones corresponding to the special words plus the silence symbol, i.e., _sil_. The division is done in this way since it is a Kaldi practice, where silence, noise and vocalized-noise phones are classified as ”silence” phones, and all phones representing traditional phonemes as ”nonsilence” phones.

Each corpus had a slightly different initial setup relative to the creation of the mentioned files. For SPEECHDAT there are some special words used to represent noise of some kind, or silence, that are between rectangular brackets in original transcriptions, e.g., [riso], or are represented in the form of a tilde or an asterisk. All these special notations are mapped to special words, mentioned above: _ehm_hmm_, _inhale_, _laugh_, _nsnoise_, _spnoise_ and _sil_. All SPEECHDAT transcriptions that contain some words followed and preceded by tilde or asterisk were mapped to _nsnoise_. Tilde was mapped to _sil_ and double asterisk to _nsnoise_. This new especial words, _ehm_hmm_, _inhale_, _laugh_, _nsnoise_ and _spnoise_, except _sil_, were mapped to specific phonemes, i.e., ehm, inh, lau, nsn, spn respectively, and added to the silence phones file, as mentioned above. All other phonemes that belong to the common words are all in the nonsilence file. These new mappings are added to lexicon.txt file, which already contains all mappings for all other words. For the out of vocabulary (OOV) words that might appear, _spnoise_ is the symbol that they will be mapped to. Finally, it is essential to note that around

28 utterances, that belong to training set of SPEECHDAT, were rectified, since their transcriptions were wrong from the original audio.

Next, for ALERT, the special words that exist in transcripts: %pp, nm, hm, è, hhh and hh, were mapped to _spnoise_. It was also created the _sil_ phone. Therefore, the only phones added to the silence list were spn and _sil_. _spnoise_ was also used to be mapped for the OOV words. Subsequently, for BD-PÚBLICO, the only special phonemes created were spn and _sil_. Despite not existing in transcriptions some special notation, _spnoise_ was created to be used as a mapping to OOV words. Finally, for TRIBUS corpus, the ”silence” and ”nonsilence” phone files are a union of all phones mentioned in SPEECHDAT, ALERT and BD-PÚBLICO, i.e., the silence phones are going to be _sil_, ehm, inh, lau, nsn and spn.

For this initial setup, it is also important to note that two more files are required for each corpus: optional_silence.txt file and extra_questions.txt file. For all corpora mentioned above, optional_silence.txt only contains phone _sil_, and extra_questions.txt file is empty.

4.2.2 HMM-GMM recipe

This section will first mention how features are extracted to train the models for BD-PÚBLICO, SPEECHDAT, ALERT, and TRIBUS. Finally, the training flow to create good ASR HMM-GMM systems for each mentioned corpus, similar to WSJ Kaldi training recipe, will be described.

Acoustic features

All features extracted for training the HMM-GMM model for BD-PÚBLICO, SPEECHDAT, ALERT and TRIBUS, respectively, are MFCCs, as mentioned in section 2.3.3. The features are extracted every 10 ms with a window of 25 ms. Since no delta and delta-delta are computed in this feature extraction procedure, the final vector for each frame contains 13 coefficients instead of 12, mainly because Kaldi uses the 0^{th} coefficient, also obtained when computing the MFCCs. One of the reasons to use the 0^{th} coefficient is that it gives a slight improvement in the WER when compared to using only 12 coefficients [83]. It is also computed cepstral mean and variance statistics per speaker, for each corpus.

Training

The overall procedure used for training state-of-the-art acoustic models in European Portuguese, similar to WSJ recipe in Kaldi, contains some main steps. First, a monophone model that does not contain any contextual information about left and right phones mentioned in 4.1, is trained to create a building block for all triphone models, also described in section 4.1, which uses contextual information. After the monophone training phase, there is an alignment phase where audio will be aligned with the corresponding reference transcript, using the trained monophone acoustic model. The main goal for this alignment phase is that additional training models can use the output, i.e., alignments, to improve their parameters. Following comes triphone training, with the computation of delta and delta-delta features, mentioned in section 2.3.3, mainly because they convey richer information about the frames context [84].

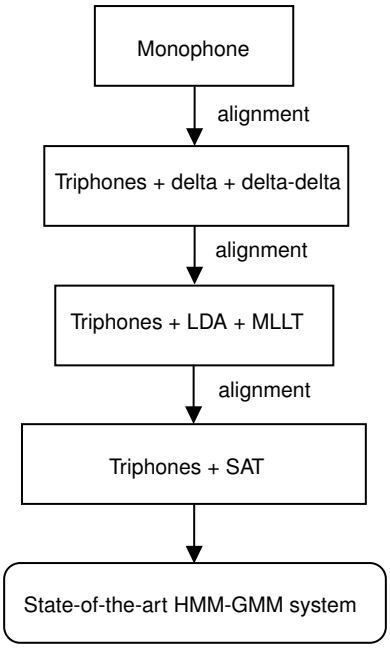


Figure 4.4: Baseline workflow of an HMM-GMM training system in Kaldi.

After performing the alignment phase, using the trained triphone acoustic model, a new triphone system is obtained using a Linear Discriminant Analysis with Maximum Likelihood Linear Transform (LDA-MLLT). LDA [85] is used for reducing dimensions of features vectors, which helps to increase accuracy [86]. MLLT [87] transforms the reduced feature space from LDA to a newly transformed space that is Gaussian distributed [86]. Following, an alignment is done using the triphone LDA-MLLT system and, at last, a triphone Speaker Adaptive Training (SAT) model is trained using the new alignments. SAT [88] is a data transform technique that normalizes all speakers to look more like each other, i.e., performs adaptation on the training set and projects training data into a speaker-normalized space. The acoustic model parameters are then computed in this new feature space, allowing the model to generalize better to unseen speakers. From this training flow, we can conclude that training a state-of-the-art HMM-GMM system involves many refinement steps. The full training scheme is depicted in figure 4.4.

All corpora (BD-PÚBLICO, SPEECHDAT, ALERT and TRIBUS) uses for each training step different parameters and number of utterances, despite having a common training workflow, as mentioned above. BD-PÚBLICO uses the 2000 shortest utterances from the original training set, to train the monophone system and 6000 utterances, from the original training set, to train the triphone system with delta and delta-delta. For the two remaining steps (triphones + LDA + MLLT and triphones + SAT), all original training set of BD-PÚBLICO is used for the training. ALERT uses the 2000 shortest utterances from the original training set to train a monophone system and a subset of 4000 utterances, from the original training set, to train the triphone system with delta plus delta-delta. The last two steps use all training set. For SPEECHDAT, the 2000 shortest utterances from the training set are used to train the monophone system and a subset of 5000 utterances, also from the training set, is used to train the second stage. For the penultimate and finale training stages, all original training set is used. Finally, for TRIBUS corpus, the 2000 shortest utterances are used to train the monophone system; a subset of 30000 utterances

from a total of 92184 (section 3.4) is used to train the second stage and, for the remaining steps, all training set is applied.

Next, all parameters used for the triphone models, such as the maximum number of leaves (numleaves) and the total number of Gaussians (totgauss) over all leaves, are depicted in the table 4.1. It was not performed an extensive search to choose the optimal parameters, except for TRIBUS. In the last training stage (triphones + SAT), the TRIBUS training set is much bigger than the other corpus. For this reason, the number of leaves and Gaussians when increased gave slightly better WER results on the validation sets. It is important to note that numleaves is the maximum number of classes the triphone tied-state tree, mentioned in section 4.4, can create while training, and numgauss is the total number of Gaussians over all leaves, where leaves with more data can get more Gaussians, i.e., Gaussian Mixture Models (GMMs).

Some more parameters were left unchanged from the WSJ Kaldi recipe. Mainly in the monophone and triphones + delta + delta-delta training, it is used a boost silence of 1.25. In the triphone LDA + MLLT system, it is used left and right context equal to 3, i.e., in the slice options. Next, related to the alignment phase before SAT training, it is used align_fmllr.sh script for all experiments instead of align_si.sh because it gives better WER performance results after training.

	numleaves	totgauss
SPEECHDAT & BD-PÚBLICO & ALERT		
Triphones + delta + delta-delta	2000	10000
Triphones + LDA + MLLT	2500	15000
Triphones + SAT	2500	25000
TRIBUS		
Triphones + delta + delta-delta	2000	10000
Triphones + LDA + MLLT	2500	15000
Triphones + SAT	5000	300000

Table 4.1: Parameters from Kaldi training setup.

Finally, it is relevant to note that decoding is performed after training each step of the HMM-GMM training workflow, depicted in figure 4.4. Furthermore, decoding is executed only to the validation set between all training stages, except for the last model (triphones + SAT), where decoding is performed to validation and test set.

4.2.3 HMM-DNN recipe

After creating strong HMM-GMM baselines for each corpus, as described in section 4.2.2, we need to obtain the alignments between input and corresponding transcriptions, for each corpus, so that we can train the HMM-DNN systems. With the auxiliary of the mentioned baselines for each respective corpus, this process of alignment will help achieve better performance results for the HMM-DNN systems. The first part of this section will explain the chain model recipe that was used to train the HMM-DNN models. This chain recipe is similar to the chain recipe that Kaldi uses for WSJ corpus [2]. It is important to note

[2] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/local/chain/tuning/run_tdnn_1i.sh

that two main experiments mentioned with more detail below were performed, where the main difference is that the second experiment used more data augmentation techniques.

Chain model

In the beginning, there is a process of feature extraction and creation of online iVectors [89], i.e., vectors in a lower-dimensional space that represent speaker or acoustic environment for each respective utterance, also known as embeddings. These feature vectors and iVectors are the input to the DNN.

First, the training data of SPEECHDAT, ALERT, BD-PÚBLICO and TRIBUS is going to be processed by a function from Kaldi that creates speed perturbed data with factors of 0.9, 1.0 and 1.1, i.e., a technique of data augmentation, mentioned in section 2.2.1. Following, MFCCs with similar characteristics as the ones mentioned in section 4.2.2, are extracted on top of the augmented training features, along with cepstral mean and variance statistics per speaker. This new data is then aligned with the corresponding transcriptions, using the baseline HMM-GMM system, mentioned in section 4.2.2. It is essential to remind that this process will be executed for BD-PÚBLICO, SPEECHDAT, ALERT and TRIBUS, respectively. In the next step, volume perturbation (using Kaldi script perturb_data_dir_volume.sh) will be applied for the training speed perturbed data. More specifically, a random scaling factor between 0.125 and 2 will be applied for each recording. After creating volume perturbation in the speed perturbed training set, cepstral mean and variance statistics and 40 MFCCs features will be computed for the new training set and the unchanged validation and test sets. Finally, using these new transformed features, online iVectors of size 100 are extracted for the training, validation and test sets, for each utterance.

A second experiment was performed with TRIBUS. Before the speed perturbed data step mentioned above, BD-PÚBLICO training data was augmented with reverberation and sounds of music, noise and speech. For creating reverberation, it was used a database of simulated and real room impulse responses, isotropic and point-source noises (RIRs) [90], with the auxiliary script reverberate_data_dir.py from Kaldi. Following, the original BD-PÚBLICO training data was augmented, resorting to a Kaldi script named augment_data_dir.py, with noise, music and babble (indistinctly or meaningless sounds) from the MUSAN corpus [91], in a separate way. After, these four new sets were combined into one, and at last, this new data was joined with the remaining training set of SPEECHDAT and ALERT, giving a total of 117351 utterances, i.e., 47552 from ALERT, 36243 from SPEECHDAT and 8389*4 from BD-PÚBLICO. The remaining steps for this experiment are the same mentioned for the other experiment mentioned above, before the speed perturbation augmentation technique. It is relevant to note that we only kept BD-PÚBLICO augmented data instead of adding the clean version. The goal was to replace the clean utterances from read speech by the ”noisy” versions to generalize better for ALERT and SPEECHDAT, which have worse performance results in general.

For each corpus, the DNN architecture used is the one used in WSJ chain recipe from Kaldi: TDNN-F, briefly mentioned in section 4.1. All architecture, training and decoding setup are almost the same as in WSJ chain recipe, except for some different subtleties mentioned below. SPEECHDAT architecture contains 13 layers. The first layer with 1024 nodes contains a ReLU (section 2.2.1) followed by batch normalization (section 2.2.1), and the remaining ones are TDNN-F layers, where each contains 1024 nodes

with a bottleneck dimension of 128. From the second to the fourth layer, the time stride used is 1. The fifth layer contains a stride of 0 and the remaining ones have a stride of 3. For ALERT and BD-PÚBLICO, 9 is the number of layers used, the same ones as SPEECHDAT, where the only difference is that only the first 9 of SPEECHDAT are used. At last, for the first and second TRIBUS experiment, the TDNN-F architecture used 12 layers, where the first 12 are the same as the first 12 used for SPEECHDAT. All mentioned layers use the L^2 regularizer, mentioned in section 2.2.1, with a value of 0.01.

Relative to the training configuration "num-jobs-initial" and "num-jobs-final" parameters were set to 1, and "use-gpu" to wait. Also, SPEECHDAT and ALERT were trained for 10 epochs, since it gave better results compared to training with more epochs, whereas BD-PÚBLICO and the two experiments using TRIBUS were trained for 15 epochs. Additionally, the decoding stage was performed for the validation set, after training each HMM-DNN model and, only when the best model was achieved, decoding was also executed for the test set.

4.3 Results

In this section, all results relative to each experiment, mentioned in 4.2.2 and 4.2.3, are presented. Each table contains the results for the training of HMM-GMM and HMM-DNN models, for each corpus: BD-PÚBLICO, SPEECHDAT, ALERT and TRIBUS. Associated to the intermediate steps used to train the HMM-GMM models, mentioned in 4.2.2, there is a table that shows results for every corpus, relative to validation and test sets, using the WER metric described in 2.3.1. Also, as mentioned in section 4.2.2, test set results exist only for the last model, i.e., triphones + SAT system.

4.3.1 HMM-GMM

The results regarding intermediate steps for obtaining a robust baseline system with Kaldi, for ALERT, SPEECHDAT and BD-PÚBLICO are shown in table 4.2. We can observe that the HMM-GMM baseline of BD-PÚBLICO, from a read speech domain, has better WER than the baselines of SPEECHDAT and ALERT, mainly because ALERT contains spontaneous speech and SPEECHDAT contains telephone speech, as mentioned in chapter 3. BD-PÚBLICO achieved 5.96% of WER on the test set, while SPEECHDAT and ALERT achieved on their respective test sets, 14.80% WER and 20.10% WER, respectively.

For the TRIBUS corpus, the results are depicted in table 4.3. It is important to remind, from section 3.4, that, despite the training set being a union of SPEECHDAT, BD-PÚBLICO and ALERT training sets, for validation and test set the report is done individually, i.e., the validation and test set is evaluated for each corpus, using its respectively language model. It is also worth mentioning that the pronunciation models used for training and decoding ALERT and BD-PÚBLICO, respectively, are different from those used for decoding with TRIBUS, as discussed in 3.4. For this reason, the overall performance can degrade because the dictionaries used for TRIBUS have no more than one pronunciation for the same word. In contrast, the other pronunciation dictionaries have different pronunciations for the same written

	WER (valid)	WER (test)
SPEECHDAT	valid	test
Monophone	58.12	-
Triphones + delta + delta-delta	38.97	-
Triphones + LDA + MLLT	32.51	-
Triphones + SAT	16.02	14.80
BD-PÚBLICO	valid	test
Monophone	21.45	-
Triphones + delta + delta-delta	7.91	-
Triphones + LDA + MLLT	6.27	-
Triphones + SAT	6.06	5.96
ALERT	valid	test
Monophone	53.53	-
Triphones + delta + delta-delta	27.51	-
Triphones + LDA + MLLT	20.52	-
Triphones + SAT	19.27	20.10

Table 4.2: SPEECHDAT, BD-PÚBLICO and ALERT WER percentages for HMM-GMM models.

word.

	WER (valid)	WER (test)
SPEECHDAT	valid	test
Monophone	39.54	-
Triphones + delta + delta-delta	10.57	-
Triphones + LDA + MLLT	8.48	-
Triphones + SAT	8.42	13.49
BD-PÚBLICO	valid	test
Monophone	58.51	-
Triphones + delta + delta-delta	14.46	-
Triphones + LDA + MLLT	11.28	-
Triphones + SAT	10.21	11.78
ALERT	valid	test
Monophone	66.72	-
Triphones + delta + delta-delta	39.09	-
Triphones + LDA + MLLT	33.96	-
Triphones + SAT	33.26	34.89

Table 4.3: TRIBUS WER percentages for HMM-GMM models.

From the results of table 4.3, and comparing with the ones from table 4.2, we can observe that TRIBUS improves the WER of SPEECHDAT, a decreasing of 1.31% WER. Despite this small improvement on the test set of SPEECHDAT, BD-PÚBLICO and ALERT degrade their performances overall. On the respective test sets, ALERT increases the WER by 14.79%, and BD-PÚBLICO increases the WER by 5.82%. This results can be a consequence of downsampling all ALERT data and BD-PÚBLICO to 8kHz, as mentioned in section 3.4.

4.3.2 HMM-DNN

The HMM-DNN training results for ALERT, SPEECHDAT and BD-PÚBLICO are present in table 4.4. When comparing results from table 4.4 with the ones from table 4.2, we can conclude that the replacement of GMM with DNN creates a significant impact in the WER performance. From table 4.4, we can see that ALERT (from 20.10% to 8.70% WER in the test set) and SPEECHDAT (from 13.49% WER to 6.48% WER in the test set) still got the higher WERs, for the same reasons as mentioned above, for the HMM-GMM results. Also, BD-PÚBLICO went from 5.96% WER in the test set to 3.31% WER.

	WER (valid)	WER (test)
SPEECHDAT	valid	test
HMM-TDNNF	7.05	6.48
BD-PÚBLICO	valid	test
HMM-TDNNF	3.47	3.31
ALERT	valid	test
HMM-TDNNF	8.89	8.70

Table 4.4: SPEECHDAT, BD-PÚBLICO and ALERT WER percentages for HMM-DNN models.

The results for TRIBUS first experiment are depicted in table 4.5. When compared to the results from table 4.4, we can observe that the only improvements happened for SPEECHDAT and BD-PÚBLICO. SPEECHDAT, on the test set, improved from 6.48% to 4.86% WER, and BD-PÚBLICO improved from 3.31% to 3.04% WER on the test set. ALERT degraded WER performance on the test set by 0.95%.

	WER (valid)	WER (test)
SPEECHDAT	valid	test
HMM-TDNNF	2.49	4.86
BD-PÚBLICO	valid	test
HMM-TDNNF	2.56	3.04
ALERT	valid	test
HMM-TDNNF	9.69	9.65

Table 4.5: TRIBUS WER percentages for first experiment with HMM-DNN models.

At last, the results for TRIBUS second experiment are depicted in table 4.6. It is worth remembering that this second experiment uses an augmented version of BD-PÚBLICO with reverberation, sounds of music, noise and speech, as mentioned with more detail in section 4.2.3. We can observe from results that there are no improvements when compared to the first TRIBUS experiment. The WER for every corpus gets slightly worse.

	WER (valid)	WER (test)
SPEECHDAT	valid	test
HMM-TDNNF	2.57	5.05
BD-PÚBLICO	valid	test
HMM-TDNNF	2.63	3.08
ALERT	valid	test
HMM-TDNNF	9.86	9.91

Table 4.6: TRIBUS WER percentages for second experiment with HMM-DNN models.

4.4 Discussion

In this chapter, we have described how to create a robust HMM-GMM baseline system and a good HMM-DNN system, using Kaldi, for the European Portuguese corpora described in chapter 3.

The results from section 4.3.2, show that the replacement of GMMs by DNNs have a significant impact on the WER performance, despite not being close to human error performance. We can also observe that training with TRIBUS has no great improvements compared to training ALERT separately, except for SPEECHDAT and BD-PÚBLICO. It is also interesting to note from results that the HMM-GMMs models do not decrease WER when training with more data, as opposed to the HMM-DNN models, which seem to benefit for the increase of data.

A more relevant result from ALERT shows how the performance of state-of-the-art systems evolves over the years. In 2010, ALERT achieved a WER percentage of 23.5 in the work [92]. Now, using the same language model and the same pronunciation model, the WER percentage decreased to 8.70, as observed in 4.4. A difference of 14.8 WER percentage.

Despite the big improvements made using state-of-the-art HMM-DNN systems, there are some limitations and complexity associated with the training and the decoding of these hybrid systems:

- **Stepwise rectification:** in order to build an accurate HMM-DNN model, it is required to train many modules mentioned in section 4.2.2. First, we have to build an HMM-GMM system to obtain the phonetic alignments and the tied-state HMM structure. Despite being relatively ”easy” to run all these modules using Kaldi, it is challenging to code and optimise all of them from the beginning.
- **Linguistic data:** a pronunciation dictionary is required to factorise language models and acoustic models, usually created by experts. For this reason, these dictionaries require too much effort, time and are prone to human error.
- **Complicated process for decoding:** decoding is done through integrating all modules, i.e., acoustic model, language model and pronunciation model. Despite being, often, efficiently handled by finite-state transducers, the implementation and optimisation of these are very complicated.
- **Incoherence in optimisation :** the acoustic, language and pronunciation models are created and optimised separately with different goals. Since each module is not trained to match the others, there could be incoherence in optimisation.

- **Assumption about conditional independence:** traditional ASR models use Markov assumptions, i.e., conditional independence assumptions, as mentioned in section 2.1. In real-world problems, this is not what happens, which leads to model misspecification, since these models do not account for everything they should.

The next chapter will explore a different approach for ASR named end-to-end speech recognition to address all the above issues.

Chapter 5

End-to-end architectures

The primary intent of this chapter is to describe the creation of state-of-the-art end-to-end systems for each corpus mentioned in chapter 3. We start by describing a brief overview of some main key concepts behind end-to-end architectures in section 5.1. Afterwards, in section 5.2, we perform end-to-end training for each data set, individually, discussing the architectures and algorithms used for training. Next, in 5.3, we relate the results of training the end-to-end architectures and compare it with the HMM-DNN trained systems. After creating state-of-the-art end-to-end ASR systems for European Portuguese, we report different experiments and results in section 5.4, to improve even more the end-to-end ASR systems. We also explore speaker invariant approaches and propose a novel architecture that incorporates neural Turing machines (NTMs). Finally, in section 5.5, we discuss some conclusions regarding end-to-end ASR models.

5.1 Related work

End-to-end speech recognition systems completely remove the HMM out of the equation and can typically solve the mathematical statement in equation 2.52 without the factorization into two different problems: acoustic modeling and language modeling, as attended in beginning of section 2.3.1. Most importantly, an end-to-end system directly maps the audio input features sequence to a sequence of words, subwords or letters (figure 5.1), therefore it has a much simpler training pipeline that shortens training time and decoding time, when compared to HMM-based approaches, mentioned in section 4.1. These systems have absence of hand-engineered knowledge (e.g., SAT, MLLT and LDA), so when training set is small it might do very poorly compared to HMM-based approaches, because of the overfitting that happens with the training data set, due to few available data. However, for supervised training and when train set is considerably large, the end-to-end model can have potential to do very well, if conceived with the right network architecture. Also, these end-to-end models only execute soft alignment, i.e., each audio frame corresponds to all possible states with a certain probability distribution, instead of a mandatory and unique correspondence (hard alignment) [14].

There are two main architectures in the end-to-end literature that are going to be described in follow-

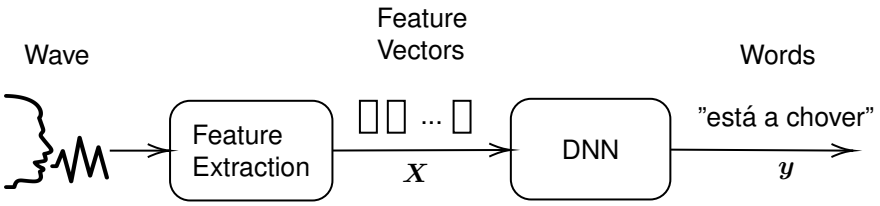


Figure 5.1: End-to-end architecture.

ing sections: *connectionist temporal classification* (CTC) based and *attention-based encoder-decoder* architectures.

Connectionist temporal classification

Proposed in [14], CTC was the first and nearest successful thing to an end-to-end system (outputs were only phonemes and it required language model) that modeled time-domain features and worked out problems where the alignment between input speech and target sentences is unknown. Being able to solve this kind of problems is critical in end-to-end, since it enables training in DNNs with the auxiliary of an objective function. With this achievement from CTC, there is no need to pre-segment, i.e., segment and align, training data and to post-process the output, in order to obtain the final transcriptions, i.e., the network directly outputs the target transcriptions.

Given an input sequence $\boldsymbol{x} = (x_1, x_2, ..., x_T)$ of length T, the RNN outputs the sequence $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_T)$. Each entry $\boldsymbol{y}_t = (y_t^1, y_t^2, ..., y_t^{|L|+1})$ is a probability distribution vector computed by a softmax operation whose dimension is the number of elements in the language L (e.g., the number of phonemes), plus the blank label, i.e., $\boldsymbol{y}_t \in \mathbb{R}^{|L|+1}$. It is also relevant to note that y_k^t indicates the probability that the output at time step t is label k. Considering $L^{'} = L \cup \{\text{blank symbol}\}$, $L^{'T}$ denotes the set of all sequences of length T defined on $L^{'}$, and:

$$p(\pi|\boldsymbol{x}) = \prod_{t=1}^{T} y_{\pi_t}^t(x) \ , \forall \pi \in L^{'T} \tag{5.1}$$

denotes the conditional probability of any sequence π, denoted as path, present in set $L^{'T}$, where π_t denotes the label at position t of path π. It is relevant to note that from equation 5.1 the outputs are all independent from each other, i.e., at any time step, the label chosen does not affect the label distribution at other time steps. Meanwhile, the computation of value y_k^t is affected by the speech context information, both from history and future directions of the network. Thereby, it can be deduced that CTC only models the acoustic model and not the language model.

From equation 5.1, we can also note that the length of the input speech is equal to the length of the output path, and, as discussed above, this is not what happens in reality. Therefore, a sequence-to-one design, $\mathcal{B}$, mentioned in section 2.2.3, must be applied, in order to get a shorter sequence of labels, since the transcription is always shorter or equal when compared with the speech acoustic features sequence. Let $L^{\leq T}$ denote the set of possible labellings with length less than or equal to T, and the sequence-to-one mapping as $\mathcal{B} : L^{'T} \rightarrow L^{\leq T}$. This mapping simply removes all repeated labels first,

and next, all blanks from the paths, e.g., $\mathcal{B}$(ba-naa-na-) $= \mathcal{B}$(-bb-aaa-na-na) $=$ banana.

Given $\boldsymbol{l}$, that corresponds to the target label sequence, we can calculate the conditional probability:

$$p(\boldsymbol{l}|\boldsymbol{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\boldsymbol{l})} p(\pi|\boldsymbol{x}), \tag{5.2}$$

where $\mathcal{B}^{-1}(\boldsymbol{l})$ corresponds to all set of paths in $L^{'T}$ that map to labelling $\boldsymbol{l}$. The value of $p(\pi|\boldsymbol{x})$ is calculated using equation 5.1. Since 5.2 is differentiable, gradient backward propagation algorithm can be applied to train the network. Equation 5.2 is computed using the forward-backward algorithm mentioned in section 2.1, since it is difficult to figure out how many paths from $L^{'T}$ are in $\mathcal{B}^{-1}(\boldsymbol{l})$.

An approach closer to an end-to-end system was proposed in [10]. More specifically, a CTC-based ASR system was created, which directly outputs character sequences given an input speech. The main difference from the original CTC is that a new objective function based on CTC, that drives the network to a direct optimization based on WER, depicted in equation 2.54, is being used. In a nutshell, the network is learning how to spell words. The architecture used was a deep bidirectional LSTM, BLSTM, stated in section 2.2.3. This bidirectionally combined with LSTMs, allows to establish long term range context both in history and future directions, which may enable the network to learn a little bit of the language model. In this work [10], the CTC-based system combined with a language model performs similarly to a complex state-of-the-art HMM-DNN system.

We can conclude that CTC is an objective function mainly applied to RNNs architectures. The development of this new technology is outstanding for LVCSR systems and eliminates the need to align data. One hitch of CTC is that it requires a language model, as mentioned above, since the label outputs are independent of each other, which deviates from the end-to-end principle. CTC mainly focus on how to create powerful acoustic models, hence a language model is still needed. Work made in [93] also shows that a language model is essential to attain good performance at word level, using a CTC-based approach. In the following section, a totally different approach that works without the need to have a language model will be discussed.

Attention-based

The attention end-to-end encoder-decoder architecture first emerged in the context of neural machine translation [56], although that the concept of attention first appeared in [94]. As mentioned in 2.2.3, the limitation of the fixed size vector context $\boldsymbol{C}$ was solved by encoding a sequence of vectors, instead of only one, and an attention mechanism was created in order for the decoder, at each output step, assign different weights to each vector in the sequence. This novel method solved the problem of having to encode input text information into a fixed size vector, thereby permitting good encoding effect on long sentences. Shortly after, the speech recognition community started applying attention mechanisms in end-to-end systems [16, 17], where in [16] the model achieved 17.6% phone error rate (PER) of TIMIT corpus, as opposite to the 23% PER of the first modern HMM-DNN system, mentioned in section 4.1. It is also interesting to note that the best PER on TIMIT, using only neural networks and known at the

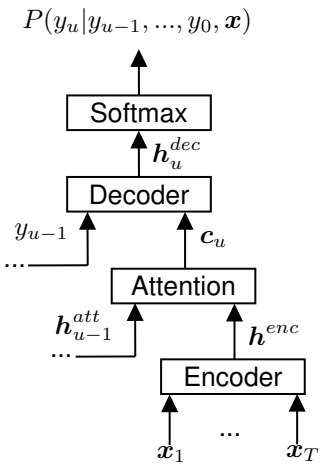


Figure 5.2: Attention-based architecture. Adapted from [68].

writing of this work, is 8.3% [13].

Attention-based architectures can be split into three distinct parts: encoder, attention and decoder as observed in figure 5.2. The encoder plays the role of the acoustic model, which is the same as in CTC-based and HMM-DNN hybrid systems. The attention part plays the role of doing alignment and the decoder assumes the equivalent role of a language model, by predicting each label as a function of previous ones and the input.

The combination of encoder and attention modules create a delay problem in all learning process of the network, given that the attention unit needs to wait for all encoding sequence result before it can start working, as opposite to CTC-based systems. Furthermore, if the encoder does not decrease the sequence size, it will have a result sequence way longer when compared to the target sequence. This imposes two problems: a bigger sequence produced after encoding translates into a bigger processing time for the attention module, thus delaying even more all processes. Secondly, since the input speech is bigger than the transcription and, without excluding some of the information created by the encoder, there will be a lot of superfluous information for the attention component to process. For the purpose of reducing the length of the sequence, Listen, Attend and Spell (LAS) in 2015 [16] adopted a 4-layer RNN to be used by the encoder in a pyramid likewise structure. Since then, most works adopted this optimization [95] and other techniques were created for the same purpose of reducing the sequence produced by the encoder [96]. The network structure of the encoder is also getting increasingly complicated, in order to improve its performance. This is reflected in the depth of the networks, that has been growing ever since - from 3 layers [17, 97] to 15 layers [11].

An attention mechanism does in reality what the word really means. In the context of neural machine translation, if we are trying to convert a long French sentence into a Portuguese one, we humans need to read some part of the sentence in French and then, translate a few number of words to Portuguese, and repeat all process until we reach the end of the translation. It becomes hard if we start by reading a long sentence (e.g., more than 35 words) and then try to translate all at once - that was what the encoder-decoder network, with no great success, was trying to do before using the attention mechanism. Attention mechanism simulates what we humans think we do, by learning what part of the French

sentence is relevant to translate the first Portuguese word, then what part of the sentence is important to translate the second Portuguese word, and so forth.

There are three types of *soft attention* mechanisms, i.e., can be trained end-to-end with back propagation, depicted in equation 5.3, that use distinct information to compute the attention weights, α_u, and therefore, establish different traits: *content-based*, *location-based* and *hybrid attention* mechanisms. The content-based computes a similarity score between the last hidden state, $\boldsymbol{h}_{u-1}^{att}$, and each produced frame of encoder, $\boldsymbol{h}^{enc}$; the location-based between the last hidden state, $\boldsymbol{h}_{u-1}^{att}$, and the previous weights, α_{u-1}, and finally, the hybrid computes a similarity score through the last hidden state, $\boldsymbol{h}_{u-1}^{att}$, the last attention weights, α_{u-1}, and the input feature vector, $\boldsymbol{h}^{enc}$, from the encoder.

$$\alpha_u = \begin{cases} Attend(\boldsymbol{h}_{u-1}^{att}, \boldsymbol{h}^{enc}) & \text{, content-based} \\ Attend(\boldsymbol{h}_{u-1}^{att}, \alpha_{u-1}) & \text{, location-based} \\ Attend(\boldsymbol{h}_{u-1}^{att}, \alpha_{u-1}, \boldsymbol{h}^{enc}) & \text{, hybrid} \end{cases} \tag{5.3}$$

These three different types of attention mechanisms, described above, must be used correctly to specific problems. For example, in machine translation tasks, it might happen that two adjacent words in the final result could be distant from each other in the original sentence to be processed. In speech recognition this is not the case, since two adjacent speech frames correspond to the two adjacent characters in the transcription result, and this problem is addressed by using either a location-based or a hybrid attention mechanism. Another issue is that if a portion of speech has already been paired to a certain label through attention, when we identify posterior labels, this portion should be not considered and must be excluded. This is something that must happen, because one output label may appear many times in different locations in the target transcription and we only want that output labels to match to their corresponding speech portions –monotonic alignment [98]. A proposal to this monotonic problem was introduced in [98] where a penalty term was introduced to the training objectives. Also it was proposed a hybrid CTC-attention architecture in [67] where the CTC enforces the encoder of the attention-based system to learn a monotonic alignment. These two problems mentioned above do not exist in CTC-based systems, since CTC implicitly works that way when doing soft alignment between input and output sequences.

Finally, the last module in attention-based architectures named decoder is commonly an RNN network that acquires context information computed by the attention mechanism and then decodes it to get the final transcription. It is important to note that labels from previous step are fed into the decoder at next step to predict $P(\boldsymbol{y}_u|\boldsymbol{y}_{u-1}, .., \boldsymbol{y}_0, \boldsymbol{x})$, as depicted in figure 5.2. This is what allows the decoder to act as a language model.

Future directions on end-to-end ASR architectures

There exists other main architectures for end-to-end ASR systems named *RNN-transducer* (RNN-T), proposed at [15], hybrid CTC-attention [67, 99–101] and transformers, first proposed in machine

translation task [102] and applied successfully into end-to-end ASR by replacing RNNs [103, 104].

New problems are always emerging and solutions for better performance proposed in the ASR field [105]. Nevertheless, attention-based encoders-decoders achieve overall better performance results when compared to CTC-based and RNN-transducer models [68]. End-to-end systems still have not surpassed HMM-DNN systems [17, 106], except for DeepSpeech [12] (because of the large amount of available training data) and Listen, Attend and Spell with SpecAugment [107], a new augmentation technique that helps to achieve state-of-the-art on significant corpora like LibriSpeech [22] and Switchboard [108]. This difference in accuracy performance, between state-of-the-art HMM-based systems and end-to-end systems, is because, the language model of end-to-end comes only from training data transcriptions, whose coverage is small. In contrast, the HMM-based systems use an additional and complex language model that provides excellent knowledge, thus relevant for better performance. Another difficulty is finding a fair tradeoff between low latency and good performance: CTC-based and RNN-T systems support stream decoding. However, their performance is bad when compared to state-of-the-art systems. Attention-based systems have a great recognition performance but have the drawback of having a long delay. Hybrid CTC-attention systems, mentioned above, try to explore both advantages from CTC and attention systems. Nevertheless, finding a fair tradeoff between decreasing latency and ensuring performance is still a research problem in the ASR community.

5.2 Experimental setup for end-to-end ASR systems

To study the full end-to-end approach without language models and pronunciation dictionaries, in European Portuguese corpora, mentioned in chapter 3, experiments were conducted using ESPnet [99]. ESPnet is a free, open-source toolkit used for end-to-end speech recognition research. More specifically, ESPnet version used for all experiments was ESPnet2 [1], in order to use the latest version of the toolkit available. The main drawback of this version is that it was still under development when this thesis was done. Also, it sill have not achieved comparable results when compared to first ESPnet version, and it is not fully optimized on each task yet. The extracted features, main architectures, training and decoding used for all end-to-end systems will be detailed with more depth in sections 5.2.1, 5.2.2 and 5.2.3. At last, in section 5.4.1, other experiments that were explored to improve the end-to-end ASR systems are going to be mentioned.

5.2.1 Acoustic features

All features extracted to train the end-to-end systems are filter bank features (mentioned in 2.3.3) with 80 dimensions, plus pitch with more 3 dimensions, giving a total of 83 dimensions for all acoustic features. These features are computed with Kaldi.

[1] https://espnet.github.io/espnet/espnet2_tutorial.html

5.2.2 Network architecture

The network used in our experiments, for each corpus mentioned in chapter 3, is similar to the one described in [99], i.e., a hybrid CTC-attention end-to-end ASR architecture. Hybrid CTC-attention was the architecture used for training all end-to-end ASR systems since it combines both advantages of CTC and attention systems. It is more feasible to configure them than transformers because they are more complex [109]. The same hybrid CTC-attention architecture that will be described with more detail below was used for all corpora, except for SPEECHDAT. For this reason, details regarding SPEECHDAT will be detailed along with the description of the main hybrid CTC-attention system. A CTC-based system and an attention-based system were also trained, only using ALERT corpus, to compare with the hybrid CTC-attention end-to-end approach. First, we will describe the main idea behind the attention-based architecture that was used, followed by a more detailed explanation of each module: *encoder*, *hybrid attention mechanism* and *decoder*. At last, a description of the hybrid CTC-attention end-to-end system will be given.

Attention-based architecture

The attention architecture contains three models: the encoder, the attention mechanism and decoder.

The encoder network

$$\boldsymbol{h}_t^{enc} = \textit{Encoder}(\boldsymbol{x}), \tag{5.4}$$

converts the input features $\boldsymbol{x}$ into a framewise hidden vector $\boldsymbol{h}_t^{enc}$. Then, we have the hybrid attention weight, similar to what is present in equation 5.3, computed as

$$\alpha_{ut} = \textit{Hybrid attention}(\boldsymbol{q}_{u-1}, \{\alpha_{u-1}\}_{t=1}^{T}, \boldsymbol{h}_t^{enc}), \tag{5.5}$$

where α_{ut} is the weight that says how much attention is going to vector $\boldsymbol{h}_t^{enc}$, in order to compute output y_u, and $\boldsymbol{q}_{u-1}$ is the last hidden state of the LSTM present in the decoder network, mentioned with more detail below. Also, it is important to note that $\boldsymbol{q}_{u-1}$ is equal to $\boldsymbol{h}_{u-1}^{att}$ from figure 5.2. After computing all weights corresponding to all framewise hidden vectors $\boldsymbol{h}_t^{enc}$, we compute a weighted summation of hidden vectors $\boldsymbol{h}_t^{enc}$ to form the hidden vector $\boldsymbol{c}_u$,

$$\boldsymbol{c}_u = \sum_{t=1}^{T} \alpha_{ut} \boldsymbol{h}_t^{enc}. \tag{5.6}$$

At last, the decoder uses the weighted summation $\boldsymbol{c}_u$ and the last output y_{u-1} to compute the new output y_u:

$$p(y_u | y_1 ... y_{u-1}, \boldsymbol{x}) = \textit{Decoder}(\boldsymbol{c}_u, y_{u-1}). \tag{5.7}$$

We will explain each module in more detail below.

Encoder network

Equation 5.4 converts the input features, 5.2.1, into a framewise vector $\boldsymbol{h}_t^{enc}$. The encoder network used consists of two initial blocks of VGG layer 2.2.2, which yields better results most of the times than the pyramidal BLSTM [99], described in 5.1. For every acoustic feature frame, we have 83x1x1 (where the first value corresponds to the dimension of the acoustic vector, the second to the frame number and the third to the number of input channels). After applying the two blocks of VGG architecture, we get 20x1x128 plus 128 of bias, giving a total of 2688 features. The number of frames is reduced approximately by a factor of 4. Following, receiving as input size the 2688 features for every reduced frame, there are 4 BLSTM layers with 1024 hidden and output units. Each BLSTM layer is followed by a linear projection layer, which receives 2048 features from the BLSTM layer (1024 of output from the backward LSTM layer and 1024 from the forward LSTM layer) and outputs 1024 features, so that they can go to the next BLSTM layer. The final output is 1024 features for every reduced frame. It is important to note that the number of input acoustic frames are all padded to have the same length so that the encoder network always has the same size.

For SPEECHDAT, the encoder architecture is different from BD-PÚBLICO, ALERT and TRIBUS in the number of hidden and output units (320), since the training set contains a lot of repeated utterances, as mentioned in section 3.3. In other words, by forcing a lower learning capacity to the encoder, the model will not overfit to the training data. Therefore it will perform better, in principle, for validation and test sets.

Hybrid attention mechanism

Hybrid attention mechanism, in equation 5.5, is decomposed as:

$$\{\boldsymbol{f}_t\}_{t=1}^{T} = \boldsymbol{K} * \boldsymbol{\alpha}_{u-1}, \tag{5.8}$$

where each $\boldsymbol{f}_t$ is going to be a vector of size 10. $*$ denotes a 1D convolution operation along axis t, with the convolution parameter $\boldsymbol{K}$, to produce the set of features $\{\boldsymbol{f}_t\}_{t=1}^{T}$.

Then, we can compute the energy value as:

$$e_{ut} = \boldsymbol{g}^T tanh(\text{LinearNN}(\boldsymbol{q}_{u-1}) + \text{LinearNNB}(\boldsymbol{h}_t^{enc}) + \text{LinearNN}(\boldsymbol{f}_t)), \tag{5.9}$$

where LinearNN is a linear layer with learnable matrix parameters and LinearNNB is a linear layer with learnable matrix and bias vector parameters. The number of output features used for the three linear networks is 320. Then, we are going to use all e_{ut} values and apply a softmax function to get the attention weight α_{ut}, so that we can compute the target output y_u:

$$\alpha_{ut} = Softmax(\{e_{ut}\}_{t=1}^{T}). \tag{5.10}$$

Decoder network

The decoder network, equation 5.7, is another RNN that computes:

$$Decoder(.) = Softmax(\text{LinearNNB}(LSTM_u)). \tag{5.11}$$

The $LSTM_u$ is conditioned on three variables:

1. the previous hidden state $\boldsymbol{q}_{u-1}$;
2. the ground truth character, y_{u-1}, which is extracted from an embedding layer with size numberOfOutputs x 1024, trained while training the full end-to-end network;
3. the attention vector $\boldsymbol{c}_u$, which is concatenated with the previous character vector, giving a vector of size 2048 as input to the $LSTM_u$ cell;

For this architecture, two LSTM cells of 1024 units were used. The new hidden state $\boldsymbol{q}_u$ is computed as:

$$\boldsymbol{q}_u = LSTM_u(\boldsymbol{q}_{u-1}, \boldsymbol{c}_u, y_{u-1}). \tag{5.12}$$

For the same reason as mentioned above, when discussing the encoder of SPEECHDAT, dropout of 0.5 was applied after each LSTM cell in the specified decoder. If no dropout was used, the decoder would overfit the language model from the training set with low linguistic variability, thus performing poorly on the other sets.

Hybrid CTC-attention network architecture

After describing the attention-based architecture, we can detail how the hybrid CTC-attention architecture works. The main idea is that the CTC and attention decoder networks share the same encoder, mentioned above. Also, when training, the CTC and attention loss are combined, to achieve more robustness and converge faster, as mentioned in 5.1:

$$Loss_{Total} = \lambda Loss_{CTC} + (1 - \lambda) Loss_{Attention}, \tag{5.13}$$

where $\lambda \in [0, 1]$. If it is 0, only CTC is being trained, and if it is equal to 1, only attention architecture is being used. In the same line of thought, if λ is between 0 and 1, except those two values, a hybrid CTC-attention system is being trained. The hybrid CTC-attention architecture, used for all experiments, is depicted in figure 5.3. As mentioned at the beginning of this section, it is also relevant to note that experiments were made to justify why hybrid CTC-attention architecture was used, instead of only using attention-based models or CTC-based models.

Following, for the training set, a difference from the end-to-end systems, when compared to the HMM-based systems, is that the non-lexical word symbols, mentioned in section 4.2.1 for the HMM-based models, are all mapped to a unique token named <noise>. Also, it is important to note that

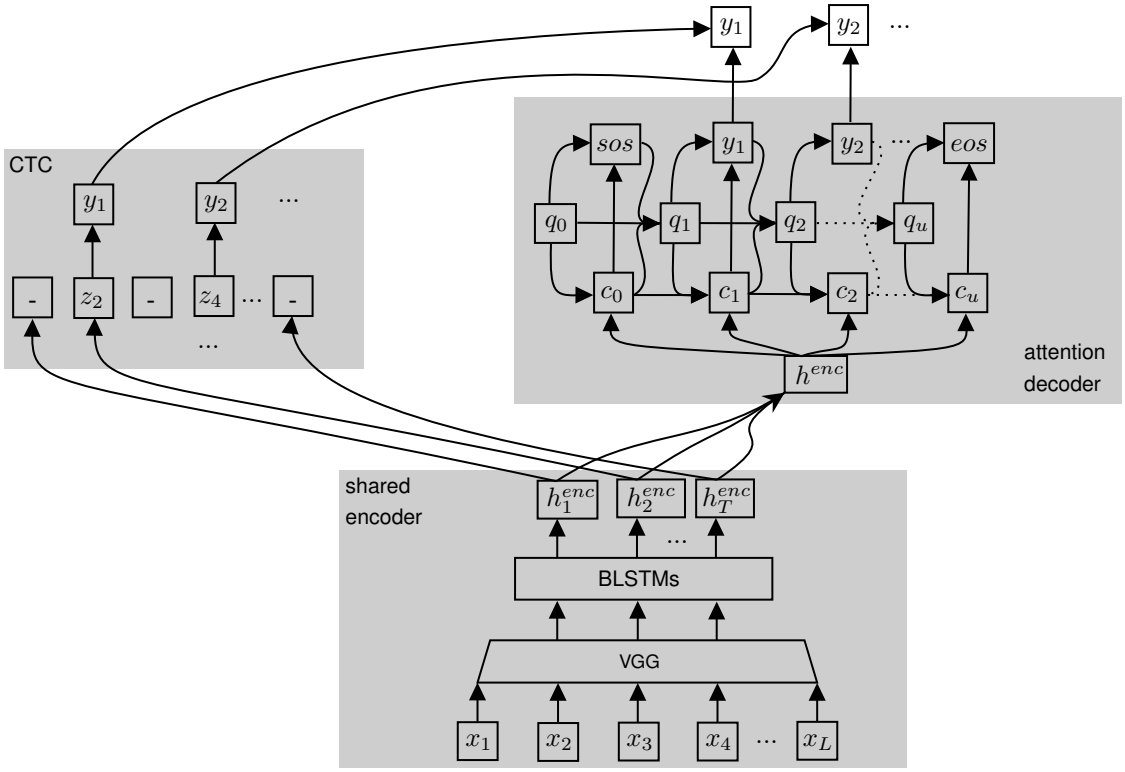


Figure 5.3: Hybrid CTC-attention model. Adapted from [99].

there are special tokens for CTC and attention systems among all other output characters that exist, respectively. CTC requires a <blank> token, as already mentioned in section 5.1, and the attention architectures requires the *start-of-sentence* and *end-of-sentence* (<sos/eos>) token. Therefore, the full hybrid CTC-attention system will have two special tokens plus an unknown token, <unk>, to map OOV symbols.

Related to the total number of output symbols for every corpus, ALERT contains 48 output symbols, including the ones mentioned above and all standard characters from the training set, plus <noise>. BD-PÚBLICO contains only 43 output symbols, including <sos/eos>, <unk> and all characters from the training set, except <noise>, since there are no special tokens in BD-PÚBLICO, as mentioned in section 4.2.1. Next, SPEECHDAT contains 45 output characters, including the special ones mentioned above, and all that appears in the training text. Finally, TRIBUS includes 48 characters (the combination of all characters mentioned for ALERT, SPEECHDAT and BD-PÚBLICO).

5.2.3 DNN training and decoding

All models were trained using a single GeForce GTX 1080 Ti. Also, they all used a learning rate of 1.0 and were trained for 30 epochs, with early stopping (patience of 4 epochs). Adadelta, an adaptive learning rate back-propagation algorithm, was the optimizer used to train all models, with a batch size of 30 and gradient clipping, mentioned in section 2.2.3, of 5. All weights were initialized using Xavier initialization. It was also used a scheduler for the learning rate, named ReduceLROnPlateau, where the factor was 0.5 and the patience 1 epoch. As for data augmentation, when first training SPEECHDAT it was

observed that by introducing speed perturbation factors (SPF) of 0.9, 1 and 1.1 and default SpecAugment from ESPnet, the results would improve by a significant margin (when SPF was introduced, WER from validation set went down from 14.4 to 8.8 and, when training with SPF plus SpecAugment, WER went down to 2.5). Therefore, the combination of SPF and SpecAugment was used as default in all training systems. Next, the weight used for CTC in training and decoding that gave better WER results for SPEECHDAT, from 0.2, 0.5 and 0.7, was 0.2. For this reason, all models were trained with a weight of 0.2 for the CTC function and a weight of 0.8 for the attention network. The same set of weights was derived from in [99]. For decoding, the weights were the same, and it was used the decoding process from ESPnet toolkit, i.e., a joint decoding by combining both attention-based and CTC scores in a one-pass beam search algorithm, as mentioned in [99]. For decoding, no auxiliary language model was used, so that we could test the full potential of the end-to-end ASR models.

At last, to keep track of each system performance during training, by using the validation set, tensorboard [2] tool from TensorFlow was used.

5.3 Results for end-to-end ASR systems

In this section, results relative to each experiment mentioned in section 5.2, are presented. First, in section 5.3.1, a comparison between CTC and attention-based systems versus hybrid CTC-attention systems will be portrayed. This results will motivate the use of hybrid CTC-attention systems over attention-based or CTC-based systems. Following, in section 5.3.2, the end-to-end ASR results for each corpus mentioned in chapter 3, using their respective architecture, mentioned in section 5.2.2, will be presented.

All results are relative to validation and test sets and presented using the metric WER, described in 2.3.1, except for the results presented in section 5.3.3. Character error rate (CER), which is computed in the same way as WER, will also be shown.

5.3.1 CTC-based and attention-based vs hybrid CTC-attention systems

The WER and CER results, from the end-to-end ASR training of CTC, attention and hybrid CTC-attention systems for ALERT, are present in table 5.1 and table 5.2, respectively. We can examine from tables 5.1 and 5.2 that hybrid CTC-attention model performs better than CTC-based or attention-based systems alone. Further, since the experiments were created by only changing λ value, mentioned in 5.2.2, we observe that attention-based system performs worse than CTC-based system for this example, as opposed to what was said in 5.1. This happens, mainly because the attention architecture is not optimized to perform better for ALERT corpus.

Next, from the experiments done, we can also note that the CTC model was trained for approximately 12 hours, whereas attention system trained within 15 hours, and the hybrid CTC-attention for 10 hours. Thus, CTC is much faster than attention alone, and the hybrid CTC-attention system is faster compared to the other two systems, i.e., CTC-based and attention-based.

[2] `https://www.tensorflow.org/tensorboard`

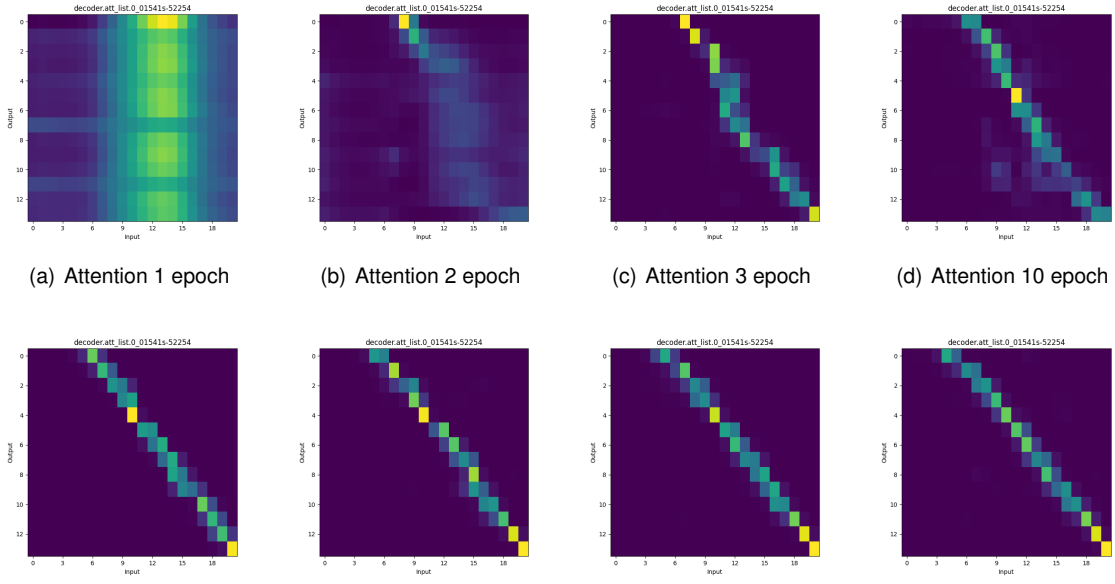
	WER (valid)	WER (test)
CTC-based	36.1	36.7
Attention-based	47.1	46.6
Hybrid CTC-attention	19.9	20.6

Table 5.1: WER percentages for ALERT.

	CER (valid)	CER (test)
CTC-based	12.1	12.2
Attention-based	30.7	29.5
Hybrid CTC-attention	8.4	8.4

Table 5.2: CER percentages for ALERT.

Following, as observed in figure 5.4 we can see that by training the hybrid CTC-attention system, monotonic alignment is learnt by the model, from the beginning, i.e., first epoch. Conversely, the attention model requires three epochs before it starts to get closer to the correct alignment. We also can see that after training for some time, at epoch 10, the attention model starts to learn wrong alignments when compared to the hybrid CTC-attention system (the CTC network forces the encoder of the hybrid CTC-attention model to learn a monotonic alignment).



(a) Attention 1 epoch (b) Attention 2 epoch (c) Attention 3 epoch (d) Attention 10 epoch

(e) CTC-attention 1 epoch (f) CTC-attention 2 epoch (g) CTC-attention 3 epoch (h) CTC-attention 10 epoch

Figure 5.4: Comparison of the speed in learning alignments between characters (y axis) and acoustic frames (x axis) between the hybrid attention system (1st row) and the hybrid CTC-attention system (2nd row) over training epochs 1 trough 3 and 10. All alignments are for the same utterance, which belongs to ALERT validation set.

5.3.2 Hybrid CTC-attention systems

The WER and CER results from the end-to-end ASR training for ALERT, SPEECHDAT and BD-PÚBLICO are shown in table 5.3 and table 5.4, respectively. SPEECHDAT, trained for 10 epochs, got the highest WER and CER compared to the other two corpora, i.e., ALERT and BD-PÚBLICO. This is because SPEECHDAT training set only contains 3622 unique utterances from a total of 36243, as mentioned with more detail in section 3.3. For this reason, the use of dropout on the decoder and the decreasing of the encoder learning capacity, mentioned in section 5.2.2, helped to decrease even further the WER and CER of SPEECHDAT.

	WER (valid)	WER (test)
SPEECHDAT	valid	test
Hybrid CTC-attention	32.2	30.4
BD-PÚBLICO	valid	test
Hybrid CTC-attention	15.3	17.5
ALERT	valid	test
Hybrid CTC-attention	19.9	20.6

Table 5.3: SPEECHDAT, BD-PÚBLICO and ALERT WER percentages.

Next, ALERT (trained for 16 epochs) got the second-highest WER and CER compared to BD-PÚBLICO, trained with 30 epochs since ALERT contains spontaneous speech and BD-PÚBLICO is only read speech. We can also verify that CER results in 5.4 are much lower than the WER results from 5.3. This means that, possibly, the end-to-end architecture is learning how to transcribe character by character better than words, considering that the main objective function is optimized at the character level and not at the word level.

	CER (valid)	CER (test)
SPEECHDAT	valid	test
Hybrid CTC-attention	11.7	11.0
BD-PÚBLICO	valid	test
Hybrid CTC-attention	4.7	5.2
ALERT	valid	test
Hybrid CTC-attention	8.4	8.4

Table 5.4: SPEECHDAT, BD-PÚBLICO and ALERT CER percentages.

The WER and CER results for TRIBUS model, which trained for 28 epochs and approximately 122 hours, are in table 5.5 and in table 5.6 respectively. We observe that WER and CER results from TRIBUS are better than the above individual results for ALERT, BD-PÚBLICO and SPEECHDAT end-to-end ASR models, from tables 5.3 and 5.4, respectively. At the WER level, TRIBUS improved SPEECHDAT from 30.4%, on the test set, to 20.0%. For BD-PÚBLICO, the test set WER decreased from 17.5% to 9.1%. Despite these great differences in WER for SPEECHDAT and BD-PÚBLICO, there was no

great difference from the end-to-end ASR baseline model of ALERT, present in table 5.3. The WER only decreased from 20.6% to 19.4%. It is also interesting to notice that the CER of the test set for ALERT, in table 5.4, stays the same when training TRIBUS, i.e., 8.4 percentage.

	WER (valid)	WER (test)
SPEECHDAT	valid	test
Hybrid CTC-attention	21.2	20.0
BD-PÚBLICO	valid	test
Hybrid CTC-attention	8.6	9.1
ALERT	valid	test
Hybrid CTC-attention	18.8	19.4

Table 5.5: TRIBUS WER percentages.

Finally, we observe that end-to-end ASR systems can, in principle benefit with the increase of data, despite coming from different source domains, i.e., telephone speech, broadcast news and read speech. Also, TRIBUS is the best end-to-end ASR model for BD-PÚBLICO, SPEECHDAT and ALERT.

	CER (valid)	CER (test)
SPEECHDAT	valid	test
Hybrid CTC-attention	8.9	8.4
BD-PÚBLICO	valid	test
Hybrid CTC-attention	2.7	2.7
ALERT	valid	test
Hybrid CTC-attention	8.5	8.4

Table 5.6: TRIBUS CER percentages.

5.3.3 Hybrid CTC-attention vs HMM-DNN systems

For this section, the main goal is to compare the best end-to-end ASR system obtained for BD-PÚBLICO, SPEECHDAT and ALERT, i.e., TRIBUS, with a strong HMM-DNN baseline obtained from chapter 4, that also used TRIBUS for training. The HMM-DNN system must also be trained on TRIBUS, in order for the comparison to be more equitable and fair. Therefore, we picked the model trained in the first experiment, mentioned in section 4.3.2. Also, it was performed scoring of the CER on the HMM-DNN test sets (using score_kaldi_cer.sh script from Kaldi) to compare the CER of both models. Only the respective test sets results, i.e., SPEECHDAT, BD-PÚBLICO and ALERT, are presented.

The WERs and CERs results for the TRIBUS HMM-DNN model, which were obtained by performing the first experiment mentioned in section 4.3.2, are depicted in table 5.7, along with the end-to-end TRIBUS WERs and CERs results.

Before comparing the hybrid CTC-attention end-to-end ASR system with the HMM-DNN system, both trained with TRIBUS, it is important to notice that the HMM-DNN system uses an in-domain language

	End-to-end		HMM-DNN	
	WER (test)	CER (test)	WER (test)	CER (test)
SPEECHDAT	test	test	test	test
HMM-TDNNF	20.0	8.4	4.86	3.26
BD-PÚBLICO	test	test	test	test
HMM-TDNNF	9.1	2.7	3.04	0.95
ALERT	test	test	test	test
HMM-TDNNF	19.4	8.4	9.65	4.33

Table 5.7: TRIBUS WER and CER percentages for the HMM-DNN first experiment, mentioned in section 4.3.2, and the end-to-end TRIBUS model.

model for each evaluated corpus, i.e., SPEECHDAT, ALERT and BD-PÚBLICO. In contrast, the end-to-end systems do not use any external language model or pronunciation dictionary. Both models also use data augmentation techniques, i.e., the hybrid CTC-attention end-to-end ASR system uses SPF (0.9, 1..0 and 1.1) and SpecAugment, while the HMM-DNN uses SPF (0.9, 1.0 and 1.1). In particular, the HMM-DNN system uses extracted iVectors for each utterance.

The first thing we notice by comparing results from table 5.7, is that the HMM-DNN systems perform much better than hybrid CTC-attention end-to-end ASR systems, for low resources – especially at the WER level. Nevertheless, the CERs of the end-to-end ASR systems start to be closer to CERs of the HMM-DNN systems. In particular, for BD-PÚBLICO, the CER from the hybrid CTC-attention end-to-end ASR system, 2.7%, is very close to the CER of the HMM-DNN system, 0.95%, where the absolute difference between the two is only 1.75%. For ALERT, the CER absolute difference between the two models is 4.07%.

Finally, when comparing the individual end-to-end ASR trained models, from table 5.3, with the individual HMM-DNN trained models, from table 4.4, for each respective corpus (SPEECHDAT, ALERT and BD-PÚBLICO), we conclude that conventional HMM-DNN systems still have a significant advantage over state-of-the-art end-to-end ASR systems, for low resource settings.

5.4 Other experiments

Beyond all experiments performed to create the end-to-end ASR systems, for the European Portuguese corpora, a few more experiments were conceived to further enhance the performance of end-to-end ASR systems, without resorting to the use of external data, such as language and pronunciation models. In the first part, 5.4.1, we will take a look at the three main experiments and associated results that were performed to create speaker invariant neural networks, and in the second part, 5.4.2, experiments and related results by applying neural Turing machines to the end-to-end ASR systems will be presented.

5.4.1 Speaker invariant ASR approach

In order to make the ASR end-to-end system more invariant to the speaker, three experiments were performed. However, before proceeding, it is relevant to note that the first two experiments were performed on SPEECHDAT, as preliminary experiments before getting the best end-to-end ASR SPEECHDAT system, and the last experiment was performed with the ALERT corpus.

The first experiment consisted of appending speaker iVectors to each acoustic feature vector, for the training, validation and test set, with the auxiliary of Kaldi programs. Following, the second experiment, similar to the first one, used iVectors extracted for each utterance instead of speaker iVectors, and at last, for the third experiment, instead of depending on the extraction of more embedding vectors, a variation of adversarial training, mentioned in section 2.2.1, was applied. Gradient reversal, as proposed in [110], was used for the adversarial training (AT). The main goal was to create a new network, detailed below, that could classify the respective speaker id. When training, this new network that shares the encoder with the hybrid CTC-attention decoder maximizes the probability of the utterance corresponding to the ground truth speaker. For the back-propagation, when arriving before the encoder, the gradient is reversed by a small factor, between 0 and 1. Thus, the encoder will be trained to create representations more invariant to the speaker ID. As a consequence, in principle, this helps the end-to-end ASR model to generalize better for unseen speakers on the test set.

The new classifier network architecture receives as input the vector of dimension 1024, an average of all vectors outputted by the encoder. A feedforward layer then transforms the averaged vector to a dimension of size 512, followed by a ReLU layer. Next, a second feedforward layer transforms the 512-dimensional vector into the same dimension, followed by another ReLU. At last, a feedforward layer transforms the 512-dimensional vector into a vector of dimension of size 1366, which is the number of speakers that exist on the ALERT training set, mentioned in section 3.1. The network is then trained to classify the respective ground truth speaker. The factor, α, used for the reversal of the gradient, mentioned above, was chosen to be computed as:

$$\alpha = \frac{2}{1 + e^{-0.07 * epoch}} - 1, \tag{5.14}$$

where $epoch$ is the current epoch of the training stage. It is essential to remind that this network is removed when applying the hybrid CTC-attention system to the validation and test sets.

Speaker and utterance iVectors results

The WERs and CERs results for the speaker iVectors, utterance iVectors and for the preliminary baseline of SPEECHDAT (a hybrid CTC-attention system), are presented in table 5.8 and table 5.9, respectively.

We can observe that neither speaker iVectors nor utterance iVectors perform better than the original baseline from the WER results. However, we can see for the CER test set performance that the utterance iVectors achieves 4.9% while the baseline only obtains 5.0% CER. Despite this result, appending speaker or utterance iVectors to the initial acoustic vectors do not significantly improve hybrid

	WER (valid)	WER (test)
Baseline	8.8	8.9
Baseline + speaker iVectors	9.6	9.7
Baseline + utterance iVectors	9.2	9.0

Table 5.8: WER percentages for SPEECHDAT - iVectors.

	CER (valid)	CER (test)
Baseline	5.0	5.0
Baseline + speaker iVectors	5.1	5.2
Baseline + utterance iVectors	5.0	4.9

Table 5.9: CER percentages for SPEECHDAT - iVectors.

CTC-attention systems performance.

Adversarial training results

The test set WERs and CERs results for the adversarial training and baseline of ALERT (a hybrid CTC-attention system) are presented in table 5.10.

	WER (test)	CER (test)
Baseline	20.6	8.4
Baseline + AT	23.2	9.7

Table 5.10: WER and CER percentages for ALERT - Adversarial training (AT).

From results, we can see that this kind of adversarial training does not improve when compared to the baseline hybrid CTC-attention system. After training several times the adversarial training system, to achieve the shown results, it was discovered that ”deep models already learn speaker-invariant representations” in the work of [18]. The mentioned work did a similar experiment for CTC-based models and found out that adversarial training for speaker invariant did not improve the baseline architecture. Finally, we can conclude that, with enough layers, the encoder of the end-to-end ASR system learns already invariant speaker representations.

5.4.2 Memory-based ASR approach

Neural Turing machines, mentioned in section 2.2.3, despite having been created in 2014, were only applied to a small number of tasks, mainly because of their difficulty to train [111]. For end-to-end ASR two known applications, from 2020, are using NTMs. The first one is from [19], where the NTM is used to store iVectors and read from them to combine with the hidden vectors of the encoder. In this work, it is essential to note that no write operation for the NTM exists, therefore this approach ignores the full potential of the NTMs. The other known work is [20], where they combine the NTM

with the decoder network, in order to improve the LM of the end-to-end model. Conversely to the first-mentioned approach that uses the NTM, our experiments use the write head. Also, they do not include any extracted iVectors. It is important to know that all experiments were created using ALERT corpus and the respective architecture, mentioned in section 5.2.2.

In the beginning, we started by combining the NTM with the hybrid CTC-attention architecture. It was shortly discovered that the improvements were small when tuning the NTM hyperparameters, but when giving more weight to the CTC module, the improvements started to increase gradually. Consequently, all future experiments that were performed only use the CTC module, i.e., the λ parameter from equation 5.13 is set to 1. Now, we will look at how the NTM was combined with the CTC-based network and after, the main results for ALERT will be presented. Before proceeding to the description of the memory-based architecture, it is important to notice that all algorithmic details about the NTM are from [111].

The architecture for the CTC ASR memory-based system is depicted in figure 5.5. For each hidden vector, $\boldsymbol{h}_t$, emitted by the encoder, the Addressing block from figure 5.5 will follow all required steps to create the weights $\boldsymbol{w}_t$, as described in section 2.2.3. At the same time, the erase vector, $\boldsymbol{e}_t$, and add vector, $\boldsymbol{a}_t$, are also extracted from $\boldsymbol{h}_t$. From this point, the memory starts to read from and write to, following the detailed processes mentioned in section 2.2.3. After the read and write updates, the read vector, $\boldsymbol{r}_t$, is concatenated with the hidden vector, $\boldsymbol{h}_t$. Next, this concatenated vector goes trough a fully connected (FC) layer, which reduces the vector dimension to the original size, i.e., 1024. This procedure is sequential for all vectors, $\boldsymbol{h}_t$, created by the encoder. Therefore, there is no significant increase of delay for the CTC-based system, as opposed to the delay created when the attention module waits for the encoder to finish all computations, as mentioned in section 5.1.

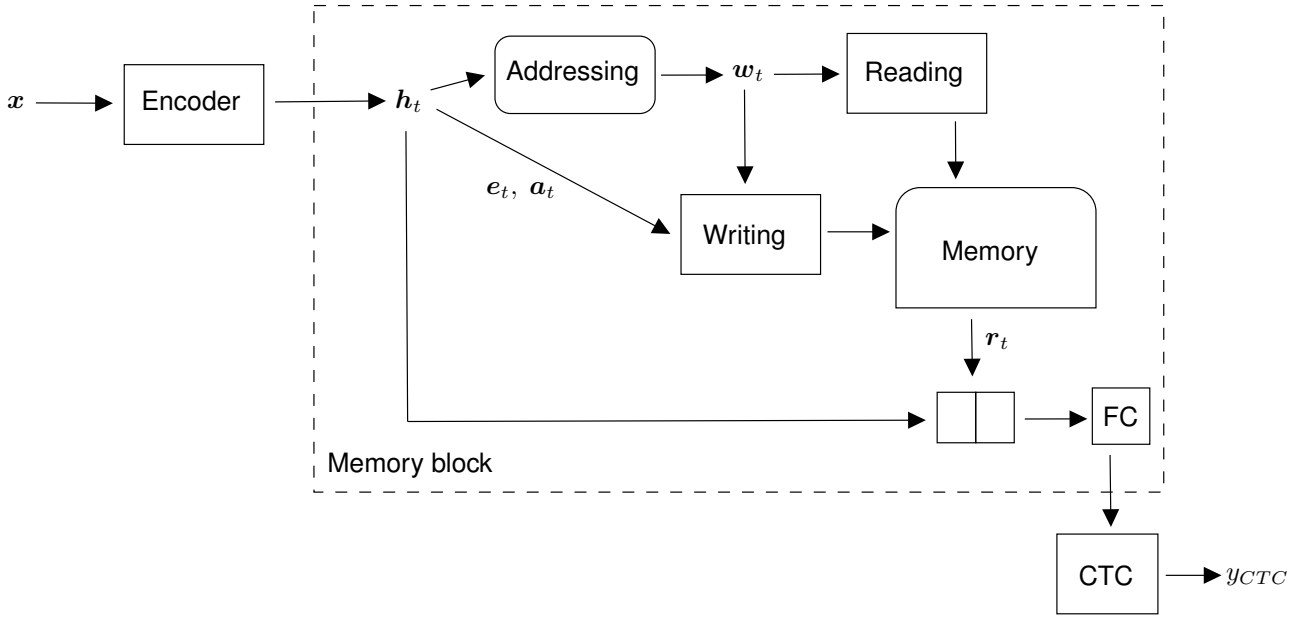


Figure 5.5: Block diagram of the CTC ASR system with a memory-based adaptation network.

The training and decoding setup is the same as mentioned in section 5.2.3, except for the patience number used to do early stopping, which is 7, and, more importantly, the weight of λ is 1 for training and

decoding, i.e., only CTC network is being used. The ESPnet2 version used is a more recent one, mainly because these experiments were performed much later than all experiments mentioned above.

Results

All WERs and CERs results are presented in tables 5.11 and 5.12, respectively. The NTM baseline architecture, which is only denoted by NTM in both tables, contains only 1 head for reading and writing, 1 of shift, 128 rows by 20 columns (128x20) and all memory entries are initialized with zeros. The information that appears between parentheses shows what is changed over the NTM baseline, mentioned above. In table 5.11, the third column corresponds to the number of training epochs for each configuration. This information replicates to table 5.12.

	WER (valid)	WER (test)	#epochs
CTC baseline	35.0	35.3	30
CTC baseline + NTM	34.1	35.1	25
CTC baseline + NTM (225x20)	33.6	33.8	28
CTC baseline + NTM (128x40)	**33.0**	**33.6**	30
CTC baseline + NTM (225x40)	34.0	34.5	29
CTC baseline + NTM (2 heads)	33.1	33.7	30

Table 5.11: WER percentages for ALERT - NTM.

	CER (valid)	CER (test)
CTC baseline	11.7	11.8
CTC baseline + NTM	11.5	11.7
CTC baseline + NTM (225x20)	11.3	11.3
CTC baseline + NTM (128x40)	**11.2**	**11.2**
CTC baseline + NTM (225x40)	11.4	11.6
CTC baseline + NTM (2 heads)	**11.2**	**11.2**

Table 5.12: CER percentages for ALERT - NTM.

We can see from table 5.11 that the CTC baseline plus the memory-based approach, using 128 rows and 40 columns, achieves the best WER on the test set, 33.6%. More importantly, we notice that the WERs of the CTC memory-based approaches always improve over the CTC baseline. For example, the CTC-based model that uses the NTM configured with 128 rows and 40 columns, decreases the WER from 35.3% to 33.6% on the test set. An absolute WER difference of 1.7%. Additionally, when observing the table with CERs, we see two models with the lowest CER. One of them is the one that achieved the best WER, mentioned above, and the other is a CTC with an NTM that uses 2 heads for reading and writing, instead of only one. Nevertheless, the best model is the CTC with an NTM that contains a memory of size 128x40.

Overall, in principle, CTC-based systems can improve the WER and CER performance if combined

with an NTM. In the near future, more configurations will be tried out, and all experiments presented in tables 5.11 and 5.12 will be replicated for the WSJ and Librispeech corpora, since, up to the best of our knowledge, there is no work done in this way.

5.5 Discussion

In this chapter, we have described how to create state-of-the-art end-to-end ASR systems, using ESPnet, for the European Portuguese corpora described in chapter 3. From experiments, we can see that WER results are not very close to the HMM-based results from section 4.2, but start to be comparable. The HMM-DNN models still have the advantage of having a language model that comprises almost all linguistics present in validation and test sets, and a pronunciation dictionary with many pronunciations for the same written words. Nonetheless, it is already impressive that the end-to-end hybrid CTC-attention systems can learn so much without any language model or pronunciation dictionary, especially the CERs that were obtained, present in section 5.4. Overall, the main problem of end-to-end ASR systems is that they do not perform well for low data resources. For example, we noticed that when training with more data, i.e., training TRIBUS model, the WER performance improved overall for BD-PÚBLICO, SPEECHDAT and a little for ALERT. For this reason, possibly a new architecture is required or, perhaps, a new unsupervised learning algorithm that could create better representations for the input data, such that it could be used to solve the supervised task, in a much better way than the modern end-to-end ASR systems.

Also, we studied and confirmed that hybrid CTC-attention models are faster than attention-based systems at learning monotonic alignments, mainly since the CTC network forces the shared encoder of the hybrid CTC-attention system to learn a monotonic alignment. Next, some experiments were performed in order to create speaker invariant neural networks, with the auxiliary of iVectors and adversarial training, but with no great success. Finally, and more notably, we developed a novel way to train CTC-based models using a memory-based approach, were the results, shown in section 5.4.2, are very promising.

Chapter 6

Conclusions

The ASR field has evolved notably since 1952, from systems that could recognize only digits to HMM-based systems that can recognize as many words as we want. More recently, full end-to-end systems inspired by the brain are being developed for the ASR field. We can already observe the improvement in ASR from a 10-year difference with the ALERT corpus. In 2010, the WER achieved in the test set was up to 23.5% [92], while now, using the same language and pronunciation models, and training data, a WER of 8.70% can be obtained, in the same test set. Most of the evolution in the field is due to deep learning new training methods (e.g., ReLU, initialization parameter algorithms, momentum and Adam), more computational power and more data. Currently, the community of deep learning and ASR is trying to not depend in the latter, by using different techniques that move away from labelled data and supervised training, e.g., semi-supervised and unsupervised learning.

The main initial goal for this project was to create state-of-the-art end-to-end systems for European Portuguese corpora on the following domains: read speech (BD-PÚBLICO), spontaneous speech (ALERT) and telephone speech (SPEECHDAT), and compare it with state-of-the-art HMM-based systems. This goal achievement encompassed the creation of hybrid CTC-attention systems for each corpus mentioned above, using ESPnet toolkit, and HMM-based systems for the same corpora using Kaldi toolkit. It was also trained a model with all corpora training data combined. This new corpus was named TRIBUS. Despite the HMM-based systems achieving better WER results compared to the hybrid CTC-attention systems, with low data resources, they still have many advantages over HMM-based models. The end-to-end method does not require HMM-GMM training for initial alignments, complex search during decoding, e.g., a WFST decoder and the creation of external lexicon and language models. The complexity and code size associated with creating these end-to-end systems are also very reduced compared to creating HMM-based systems.

Despite the better WER results for HMM-based ASR systems in low resource settings, we learned that CER results are outstanding, e.g., ALERT with TRIBUS achieves 8.4% CER on the test set domain, which is comparable with 4.33% CER, from the HMM-DNN baseline of ALERT, also using TRIBUS, on the same test set domain. We can also confirm that HMM-based and end-to-end systems learn, more easily, to model read speech than spontaneous speech in the presence of low data resources. This is

one of the disadvantages of supervised learning when data is limited. Augmentation techniques improve the WER considerably, especially, in end-to-end ASR systems, but they can not cover all examples in test set or real application.

One of the project major contributions is that it is, up to the best of our knowledge, the first work using state-of-the-art end-to-end systems for low resource European Portuguese corpora. We also learned that for end-to-end ASR systems with low data resources, especially hybrid CTC-attention models, speed perturbed data plus SpecAugment augmentation techniques are a good combination to achieve better performance. Also, we studied that, if the encoder contains many layers, the "thought vector" produced by the encoder, coined by Geoffrey Hinton [1], is already invariant to speaker particular characteristics. We also observed that appending iVectors to the input acoustic features or using adversarial training for speaker invariant, does not improve the performance of the end-to-end system. If they do, the improvements are not very significant. Also, we confirmed that hybrid CTC-attention systems perform better than pure attention-based models, since the former, with the CTC module, enforces the shared encoder to learn the right monotonic alignments between the input speech and output characters. At last, and more notably, we proposed a novel way of training CTC-based models using a memory-based system inspired by the NTM model. The improvements, when compared to a normal CTC-based system, are quite remarkable. Overall, all the objectives and research questions stated in the introduction were satisfactorily achieved.

Finally, it is important to notice that, despite this thesis was originally planned to be conducted jointly at PDMFC and INESC-ID, not much work was actually performed for the company, mainly because of the pandemic situation. For this reason, more time was devoted to experimenting with new ideas that could improve the WER of the end-to-end ASR European Portuguese models, without being dependent on external factors, like language models.

Publications

The part of the work of this thesis related with the development of the European Portuguese end-to-end system has been submitted to IberSPEECH 2020 conference, and the novel contributions using memories is planned to be submitted to INTERSPEECH 2021:

- C. Carvalho and A. Abad, "TRIBUS: An end-to-end automatic speech recognition system in European Portuguese", submitted to IberSPEECH 2020.

- C. Carvalho and A. Abad, "CTC-M: A CTC-based end-to-end ASR system with memory adaptation", in preparation for INTERSPEECH 2021.

[1] https://www.youtube.com/watch?v=fDR1I2Shw_E&list=PLhl-FVpNS-ejQQ7V8kqjZJqK-VlP5F_id&index=7

6.1 Future work

At first, more configurations for the memory-based approach applied to end-to-end CTC-based ASR will be tried out forthwith. Simultaneously, as a result of the successful results for ALERT, the memory-based experiments presented in section 5.4.2 will be replicated for English corpora and published for INTERSPEECH 2021, since, up to the best of our knowledge, there is no work done in this way.

More importantly and at last, this work provides the groundwork and suggests future work paths in end-to-end ASR with low resource settings, e.g., European Portuguese. Many important problems remain, but the main one is the *problem of low resources* within deep learning. Deep neural networks are known to require many labelled data in order to achieve state-of-the-art performance results. Nevertheless, it is known that gathering and labelling data is very time consuming and also very expensive. One way to solve this problem would be to approach the end-to-end ASR system with *unsupervised learning*. Unsupervised learning algorithms try to find relevant ”structure” in data, instead of learning to perform a specific classification or regression task. Motivated by the fact that children learn how the world works by observation and remarkably little interaction, i.e., with little supervised feedback, unsupervised learning is a promising research path for end-to-end ASR with low resources. When approaching unsupervised learning in deep learning, one of the main problems that arise is how to find a good representation that could, in principle, disentangle all relevant factors for the ASR task. The problem of ”discovering good representations” is approached with more detail in [112]. Beyond exploring good representations for ASR, one could also explore generative adversarial networks (GANs) and variations of it for end-to-end ASR. Invented in 2014 [113], GANs introduced many innovations in training deep generative models outside of the maximum likelihood framework and even outside of the classical framework of having a single objective function. Instead, GANs use multiple models trained in a game-theoretical way, i.e., zero-sum games, and each model has a different objective function. The main problem with GANs, discussed in [114], is to know whether a Nash equilibrium exists and whether the learning algorithm converges to a Nash equilibrium, and if it does so, how quickly. Despite the huge success with GANs in the image domain [114], there is still the problem of applying them to other domains, e.g., speech. Finding new training techniques or better implicit priors for the speech domain is a very compelling research topic.

Bibliography

[1] C. Pérez, Y. Campos-Roca, L. Naranjo, and J. Martín, "Diagnosis and tracking of parkinson's disease by using automatically extracted acoustic features," *Journal of Alzheimer's Disease and Parkinsonism*, vol. 6, 01 2016.

[2] Y. Jia, R. J. Weiss, F. Biadsy, W. Macherey, M. Johnson, Z. Chen, and Y. Wu, "Direct speech-to-speech translation with a sequence-to-sequence model," *Interspeech 2019*, Sep 2019. [Online]. Available: http://dx.doi.org/10.21437/interspeech.2019-1951

[3] S. Ruan, J. Wobbrock, K. Liou, A. Ng, and J. A. Landay, "Speech is 3x faster than typing for english and mandarin text entry on mobile devices," *ArXiv*, vol. abs/1608.07323, 2016.

[4] A. Ng, *Machine Learning Yearning*. Online Draft, 2017.

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, p. 1527–1554, Jul. 2006. [Online]. Available: https://doi.org/10.1162/neco.2006.18.7.1527

[6] R. Hahnloser, R. Sarpeshkar, M. Mahowald, and R. Douglas, "Digital selection and analog amplification co-exist in an electronic circuit inspired by neocortex," *Nature*, 01 2000.

[7] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[8] G. Hinton, l. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, pp. 82–97, 11 2012.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[10] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," *31st International Conference on Machine Learning, ICML 2014*, vol. 5, pp. 1764–1772, Jan. 2014.

[11] Y. Zhang, W. Chan, and N. Jaitly, "Very Deep Convolutional Networks for End-to-End Speech Recognition," *arXiv:1610.03022 [cs]*, Oct. 2016, arXiv: 1610.03022. [Online]. Available: http://arxiv.org/abs/1610.03022

[12] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," *CoRR*, vol. abs/1412.5567, 2014. [Online]. Available: http://arxiv.org/abs/1412.5567

[13] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," 2020.

[14] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 369–376. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143891

[15] A. Graves, "Sequence Transduction with Recurrent Neural Networks," *arXiv:1211.3711 [cs, stat]*, Nov. 2012, arXiv: 1211.3711. [Online]. Available: http://arxiv.org/abs/1211.3711

[16] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, Attend and Spell," *arXiv:1508.01211 [cs, stat]*, Aug. 2015, arXiv: 1508.01211. [Online]. Available: http://arxiv.org/abs/1508.01211

[17] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-Based Models for Speech Recognition," *arXiv:1506.07503 [cs, stat]*, Jun. 2015, arXiv: 1506.07503. [Online]. Available: http://arxiv.org/abs/1506.07503

[18] Y. Adi, N. Zeghidour, R. Collobert, N. Usunier, V. Liptchinsky, and G. Synnaeve, "To reverse the gradient or not: An empirical comparison of adversarial and multi-task learning in speech recognition," *CoRR*, vol. abs/1812.03483, 2018. [Online]. Available: http://arxiv.org/abs/1812.03483

[19] L. Sarı, N. Moritz, T. Hori, and J. L. Roux, "Unsupervised speaker adaptation using attention-based speaker memory for end-to-end asr," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 7384–7388.

[20] D. Lee, J.-S. Park, M.-W. Koo, and J.-H. Kim, "Language model using ring machine based on localized content-based addressing," *Applied Sciences*, vol. 10, no. 20, p. 7181, 2020.

[21] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "Darpa timit acoustic phonetic continuous speech corpus cdrom," 1993.

[22] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 5206–5210.

[23] D. B. Paul and J. M. Baker, "The design for the wall street journal-based csr corpus," in *Proceedings of the Workshop on Speech and Natural Language*, ser. HLT '91. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 357–362. [Online]. Available: https://doi.org/10.3115/1075527.1075614

[24] J. Baker, “The dragon system–an overview,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 24–29, 1975.

[25] A. Waibel and K. Lee, *Readings in Speech Recognition*. Elsevier Science, 1990. [Online]. Available: https://books.google.de/books?id=yjzCra5eW3AC

[26] 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*, vol. 39, pp. 1–38, 1977. [Online]. Available: http://web.mit.edu/6.435/www/Dempster77.pdf

[27] F. B. Fitch, “Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133.” *Journal of Symbolic Logic*, vol. 9, no. 2, p. 49–50, 1944.

[28] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[29] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[30] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[31] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, Dec. 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02551274

[32] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 5, pp. 157–66, 02 1994.

[33] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[34] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV ’15. USA: IEEE Computer Society, 2015, p. 1026–1034. [Online]. Available: https://doi.org/10.1109/ICCV.2015.123

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Representations by Back-Propagating Errors*. Cambridge, MA, USA: MIT Press, 1988, p. 696–699.

[36] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 07 2011.

[37] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *ArXiv*, vol. abs/1502.03167, 2015.

[38] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?” 2019.

[39] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *AISTATS*, 2010.

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, Jan. 2014.

[41] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” 2015.

[42] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” 2014.

[43] Y. LeCun, *Generalization and network design strategies*. Elsevier, 1989.

[44] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[45] S. P. Singh, L. Wang, S. Gupta, H. Goli, P. Padmanabhan, and B. Gulyás, “3d deep learning on medical images: A review,” 2020.

[46] B. T. Nugraha, S. Su, and Fahmizal, “Towards self-driving car using convolutional neural network and road lane detector,” in *2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro.Mechanical System, and Information Technology (ICACOMIT)*, 2017, pp. 65–69.

[47] Y. T. Zhou and R. Chellappa, “Computation of optical flow using a neural network,” *IEEE 1988 International Conference on Neural Networks*, pp. 71–78 vol.2, 1988.

[48] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.

[49] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/6953413]

[50] B. Hammer, “On the approximation capability of recurrent neural networks,” *Neurocomputing*, vol. 31, 10 2001.

[51] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[52] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS’14. Cambridge, MA, USA: MIT Press, 2014, p. 3104–3112.

[53] D. Eck and J. Schmidhuber, “Finding temporal structure in music: blues improvisation with lstm recurrent networks,” in *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, 2002, pp. 747–756.

[54] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,” in *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer and J. F. Kolen, Eds. IEEE Press, 2001.

[55] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” *arXiv:1406.1078 [cs, stat]*, Sep. 2014, arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078

[56] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” *arXiv:1409.0473 [cs, stat]*, Sep. 2014, arXiv: 1409.0473 version: 1. [Online]. Available: http://arxiv.org/abs/1409.0473

[57] A. Graves, G. Wayne, and I. Danihelka, “ring machines,” *CoRR*, vol. abs/1410.5401, 2014. [Online]. Available: http://arxiv.org/abs/1410.5401

[58] J. Weston, S. Chopra, and A. Bordes, “Memory networks,” 2015.

[59] R. Csordás and J. Schmidhuber, “Improving differentiable neural computers through memory masking, de-allocation, and link distribution sharpness control,” 2019.

[60] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis, “Hybrid computing using a neural network with dynamic external memory,” *Nature*, vol. 538, no. 7626, pp. 471–476, 2016. [Online]. Available: https://doi.org/10.1038/nature20101

[61] C. Gulcehre, S. Chandar, K. Cho, and Y. Bengio, “Dynamic ring machine with soft and hard addressing schemes,” 2017.

[62] S. Renals and T. Hain, “Speech recognition,” in *Handbook of Computational Linguistics and Natural Language Processing*, A. Clark, C. Fox, and S. Lappin, Eds. Wiley Blackwell, 2010, ch. 12.

[63] N. Dave, “Feature extraction methods lpc, plp and mfcc in speech recognition,” *International Journal For Advance Research in Engineering And Technology(ISSN 2320-6802)*, vol. Volume 1, 07 2013.

[64] S. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, August 1980.

[65] M. Gales and S. Young, *Application of Hidden Markov Models in Speech Recognition*. now, 2008. [Online]. Available: https://ieeexplore.ieee.org/document/8187420

[66] R. Togneri, A. M. Toh, and S. Nordholm, “Evaluation and modification of cepstral moment normalization for speech recognition in additibe babble ensemble.”

[67] S. Kim, T. Hori, and S. Watanabe, “Joint ctc-attention based end-to-end speech recognition using multi-task learning,” *CoRR*, vol. abs/1609.06773, 2016. [Online]. Available: http://arxiv.org/abs/1609.06773

[68] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, “A comparison of sequence-to-sequence models for speech recognition,” in *Proc. Interspeech 2017*, 2017, pp. 939–943. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2017-233

[69] K. Rao, H. Sak, and R. Prabhavalkar, “Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer,” 2018.

[70] H. Meinedo, D. Caseiro, J. Neto, and I. Trancoso, “Audimus.media: A broadcast news speech recognition system for the european portuguese language,” in *Computational Processing of the Portuguese Language*, N. J. Mamede, I. Trancoso, J. Baptista, and M. das Graças Volpe Nunes, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 9–17.

[71] J. Neto, C. Martins, H. Meinedo, and L. Almeida, “The design of a large vocabulary speech corpus for portuguese,” in *EUROSPEECH*, 1997.

[72] A. Hagen and J. P. Neto, “Hmm/mlp hybrid speech recognizer for the portuguese telephone speechdat corpus,” in *Computational Processing of the Portuguese Language*, N. J. Mamede, I. Trancoso, J. Baptista, and M. das Graças Volpe Nunes, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 126–134.

[73] D. Caseiro and I. Trancoso, “Spoken language identification using the speechdat corpus.” 01 1998.

[74] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz, J. Silovský, G. Stemmer, and K. Vesel, “The kaldi speech recognition toolkit,” *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, 01 2011.

[75] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer, “Speech recognition with continuous-parameter hidden markov models,” in *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, 1988, pp. 40–43 vol.1.

[76] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2nd Edition)*. USA: Prentice-Hall, Inc., 2009.

[77] T. Robinson and F. Fallside, “A recurrent error propagation network speech recognition system,” *Computer Speech & Language*, pp. 259–274, 1991.

[78] P. Smolensky, *Information Processing in Dynamical Systems: Foundations of Harmony Theory*. Cambridge, MA, USA: MIT Press, 1986, p. 194–281.

[79] A. Mohamed, G. Dahl, and G. Hinton, “Deep belief networks for phone recognition,” *NIPS 22 workshop on deep learning for speech recognition*, 2009.

[80] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, “Phoneme recognition using time-delay neural networks,” *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, pp. 328 – 339, 04 1989.

[81] V. Peddinti, D. Povey, and S. Khudanpur, “A time delay neural network architecture for efficient modeling of long temporal contexts,” in *INTERSPEECH*, 2015.

[82] D. Povey, G. Cheng, Y. Wang, K. Li, H. Xu, M. Yarmohammadi, and S. Khudanpur, “Semi-orthogonal low-rank matrix factorization for deep neural networks,” in *INTERSPEECH*, 2018.

[83] F. Zheng and G. Zhang, “Integrating the energy information into mfcc,” in *INTERSPEECH*, 2000.

[84] B. A. Hanson and T. H. Applebaum, “Robust speaker-independent word recognition using static, dynamic and acceleration features: experiments with lombard and noisy speech,” in *International Conference on Acoustics, Speech, and Signal Processing*, 1990, pp. 857–860 vol.2.

[85] S. Geirhofer, “Feature reduction with linear discriminant analysis and its performance on phoneme recognition,” 2004.

[86] A. Yuliani, R. Sustika, R. Yuwana, and H. Pardede, “Feature transformations for robust speech recognition in reverberant conditions,” 10 2017, pp. 57–62.

[87] R. A. Gopinath, “Maximum likelihood modeling with gaussian distributions for classification,” in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP ’98 (Cat. No.98CH36181)*, vol. 2, 1998, pp. 661–664 vol.2.

[88] T. Anastasakos, J. McDonough, R. Schwartz, and J. Makhoul, “A compact model for speaker-adaptive training,” in *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP ’96*, vol. 2, 1996, pp. 1137–1140 vol.2.

[89] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, “Front-end factor analysis for speaker verification,” *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, pp. 788 – 798, 06 2011.

[90] T. Ko, V. Peddinti, D. Povey, M. L. Seltzer, and S. Khudanpur, “A study on data augmentation of reverberant speech for robust speech recognition,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 5220–5224.

[91] D. Snyder, G. Chen, and D. Povey, “MUSAN: A Music, Speech, and Noise Corpus,” 2015, arXiv:1510.08484v1.

[92] H. Meinedo, A. Abad, T. Pellegrini, J. Neto, and I. Trancoso, “The l2f broadcast news speech recognition system,” 01 2010.

[93] A. Y. Hannun, A. L. Maas, D. Jurafsky, and A. Y. Ng, “First-Pass Large Vocabulary Continuous Speech Recognition using Bi-Directional Recurrent DNNs,” *arXiv:1408.2873 [cs]*, Dec. 2014, arXiv: 1408.2873. [Online]. Available: http://arxiv.org/abs/1408.2873

[94] A. Graves, “Generating sequences with recurrent neural networks,” *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

[95] J. Chorowski and N. Jaitly, “Towards better decoding and language model integration in sequence to sequence models,” *arXiv:1612.02695 [cs, stat]*, Dec. 2016, arXiv: 1612.02695. [Online]. Available: http://arxiv.org/abs/1612.02695

[96] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, “End-to-End Attention-based Large Vocabulary Speech Recognition,” *arXiv:1508.04395 [cs]*, Mar. 2016, arXiv: 1508.04395. [Online]. Available: http://arxiv.org/abs/1508.04395

[97] J. Hou, S. Zhang, and L.-R. Dai, “Gaussian prediction based attention for online end-to-end speech recognition,” in *INTERSPEECH*, 2017.

[98] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio, “End-to-end Continuous Speech Recognition using Attention-based Recurrent NN: First Results,” *arXiv:1412.1602 [cs, stat]*, Dec. 2014, arXiv: 1412.1602. [Online]. Available: http://arxiv.org/abs/1412.1602

[99] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. Enrique Yalta Soplin, J. Heymann, M. Wiesner, N. Chen, A. Renduchintala, and T. Ochiai, “Espnet: End-to-end speech processing toolkit,” in *Interspeech*, 2018, pp. 2207–2211. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2018-1456

[100] Z. Yuan, Z. Lyu, J. Li, and X. Zhou, “An improved hybrid ctc-attention model for speech recognition,” 2018.

[101] Z. Xiao, Z. Ou, W. Chu, and H. Lin, “Hybrid CTC-Attention based End-to-End Speech Recognition using Subword Units,” *arXiv:1807.04978 [cs, eess]*, Sep. 2018, arXiv: 1807.04978. [Online]. Available: http://arxiv.org/abs/1807.04978

[102] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[103] M. Sperber, J. Niehues, G. Neubig, S. Stüker, and A. Waibel, “Self-attentional acoustic models,” *CoRR*, vol. abs/1803.09519, 2018. [Online]. Available: http://arxiv.org/abs/1803.09519

[104] Y. zhao, J. Li, X. Wang, and Y. Li, “The speechtransformer for large-scale mandarin chinese speech recognition,” in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 7095–7099.

[105] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, “State-of-the-art Speech Recognition With Sequence-to-Sequence Models,” *arXiv:1712.01769 [cs, eess, stat]*, Dec. 2017, arXiv: 1712.01769 version: 1. [Online]. Available: http://arxiv.org/abs/1712.01769

[106] L. Lu, X. Zhang, and S. Renais, “On training the recurrent neural network encoder-decoder for large vocabulary end-to-end speech recognition,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 5060–5064.

[107] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, “SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition,” *Interspeech 2019*, pp. 2613–2617, Sep. 2019, arXiv: 1904.08779. [Online]. Available: http://arxiv.org/abs/1904.08779

[108] J. J. Godfrey, E. C. Holliman, and J. McDaniel, “Switchboard: telephone speech corpus for research and development,” in *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, March 1992, pp. 517–520 vol.1.

[109] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang, S. Watanabe, T. Yoshimura, and W. Zhang, “A comparative study on transformer vs rnn in speech applications,” 2019.

[110] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, “Domain-adversarial training of neural networks,” 2016.

[111] M. Collier and J. Beel, “Implementing neural turing machines,” 2018.

[112] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” 2014.

[113] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *ArXiv*, vol. abs/1406.2661, 2014.

[114] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Commun. ACM*, vol. 63, no. 11, p. 139–144, Oct. 2020. [Online]. Available: https://doi.org/10.1145/3422622