Spanish Automated Speech Recognition using Deep Learning

Author:
Shweta

GitHub:
https://github.com/shwe87/spanish-asr-deep-learning

July 2020

Contents

List of Figures

List of Tables

"The capacity to learn is a gift.The ability to learn is a skill.The willingness to learn is a choice."

-Brian Herbert

Acknowledgements

We've been busy learning these past months, and by *we*, I mean, **the machine and me**. The machine learning how to convert speech to text and me, learning how to make it learn.

This was possible because of the unlimited information that can be found on the Internet, added by the Deep Learning Nano Degree that I accomplished in Udacity.

A big thanks to **TFCoop** for creating an astonishing community of people promoting the Sustainable Development Goals.

Gratitude to **Tucuvi**, for giving me this opportunity to learn and grow together. Thanks, Marcos, for your precious time.

Thanks to professor **Juan Manuel Vara Mesa**, I very much appreciate the patience you had with me.

I would like to dedicate this work to my **family**, for their everlasting love and support.

My husband, **Javier Haro**, for his unconditional support, for recharging me with courage and energy every time I felt I couldn't do this, for taking care of everything else while I shut myself from the rest of the world to finish this, and for believing in me.

Abstract

The objective of this project is to create or design a *Speech-to-Text* or *Automatic Speech Recognition* in Spanish application using *Deep Learning* so that the company **Tucuvi** can use it. Speech recognition application are perhaps, too invasive in our daily life. We use it instead of typing, we use it for searching, we use it for buying, we use it to know the weather, to know the time, to set-up the alarm, whatever unimaginable that you are thinking right now can be done with your voice. The machine are more intelligent, not year by year but day by day, traffic detection for avoiding it, face detection, movement detection, all these terms that were seen only in Hollywood movies can be found on your smartphone. Netflix suggesting you movies to watch next, and Amazon showing you products you should buy or browse at least which is totally linked with what you have been searching few seconds before.

All these fall in the category of Artificial Intelligence (AI) and particularly machine learning(ML), which is a subset of AI. Deep learning on the other hand, is a subset of ML, it is a technique used to perform ML based on human's neural network, and since the first image classifier implemented with deep learning, it has taken off with increasing number of applications. Deep comes from the number of neurons that are inside the its system. Neural networks, connected together, passing information from one another, that is similar to how human's neurons work. Deep learning methods are popular each day because it is a system which is capable of learning by itself using these neural networks.

Speech Recognition is one of the best performing application in deep learning. Although, in real-world, we can only perceive speech, we cannot see it, in mathematical terms or in physical terms it can be measured. These measurements are from what the neural networks can learn by itself by finding patterns. To find patterns, it has to be injected with as many data as possible, it will have more patterns to learn, to compare. To comprehend how to apply deep learning is a learning process itself and it is in continuous evolution. Hundreds of literature to read about it and more literature and methods for efficiency are being released even nowadays. However, there are sections that deep learning has invaded and speech recognition is one of them. Deep learning has many parameters and many options to play with and finding the optimum one would have been difficult for this project if there were no previous work to lean on. Implementation like deep speech was revolutionary in this area, so, why not inspire on those models? The method used to accomplish the goal was inspire one several previously created models and create several myself based on those, keep on trying until getting one that performed the best. Actually, more models were tested than the ones mentioned in this document, however most of the time, those models were part of the self-learning process so I didn't think convenient to mention those here.

The best model that out-performed above all the tested ones, was a deep neural network created by using two convolutional layers and six bidirectional Gated Recurrent Unit. With limited data and resources, a word error rate of 76.32% was achieved. There are methods to improve this rate, getting more data was probably the most important handicap in this project, followed by lack of Graphic Processing Unit. The main question that needs to be answered is if competing with companies like Google is worth it? Google has more data than anyone can dream of, creating a speech recognition system for research or for learning process can be fun but creating one specific for some specific proposal may not be as profitable as one could think. Is there anybody ready to beat big companies or even challenge companies like Google, Amazon, Facebook, etc. in terms of data and technology?

1

Introduction

In the twentieth century, providing “intelligence” to lifeless objects had scarcely started to emerge as an unclear idea, and was starting to take its baby steps. Today, these lifeless objects or machines aren’t some vague ideas, they are present in our daily life and these types of applications are growing day by day.

On 1950, A.M. Turing asked himself if machines could think for themselves, in his paper Computing Machinery and Intelligence [1] published on 1950, he reasons and explains that the machines can not only do what we tell them to do but also learn by themselves. In his paper, A.M. Turing is informing the world that with sufficient information (data) and reasoning, a machine could think for themselves. He was talking about **artificial intelligence**, a machine with its own "mind". Thus, A.M Turing was the first man to offer the possibility of creating or designing **Artificial Intelligence AI** as a reality.

He triggered this idea and it was not until 1986, Carnegie Melon, built the very first autonomous car. Nearly after forty years since Turing wrote his paper, as shown in the timeline represented in the figure [1.1]. Why so? Why didn’t anyone create any machine with artificial intelligence immediately after Turing’s paper? Most of the objectives described in the first section have been already implemented.

According to Moore’s law [2], electronic devices, as well as the price of the components, will

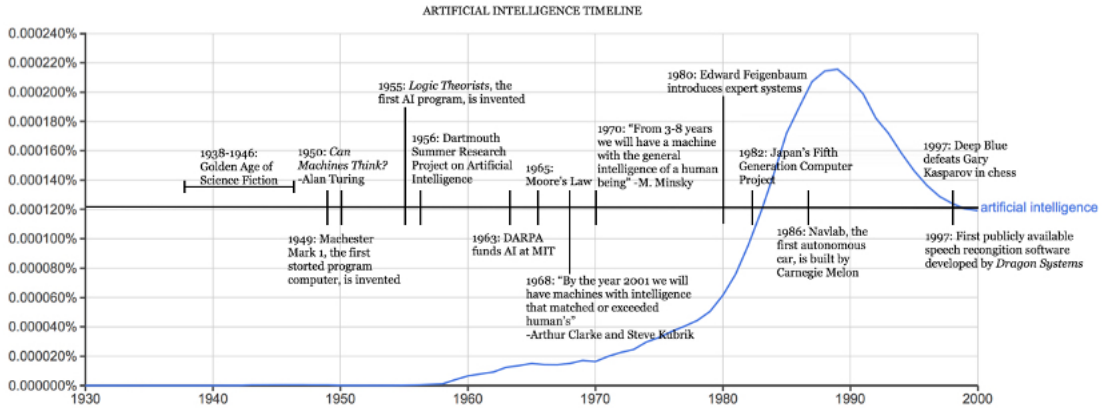
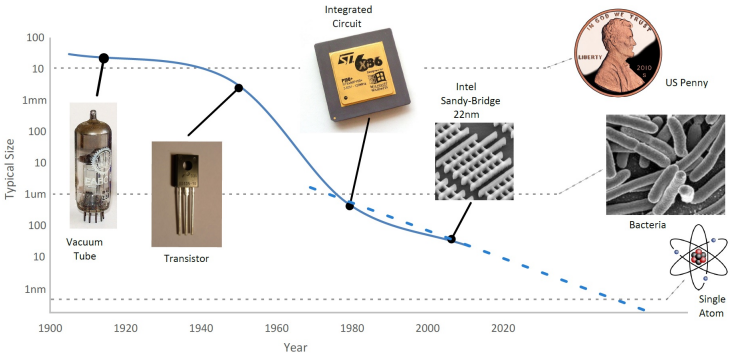
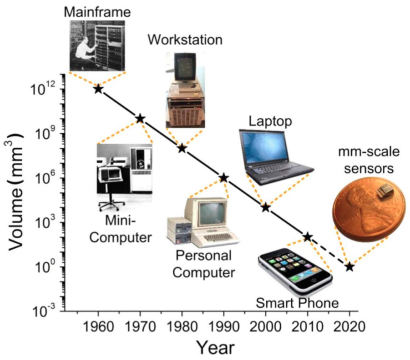


Figure 1.1: Artificial Intelligence Timeline
Source: Harvard University

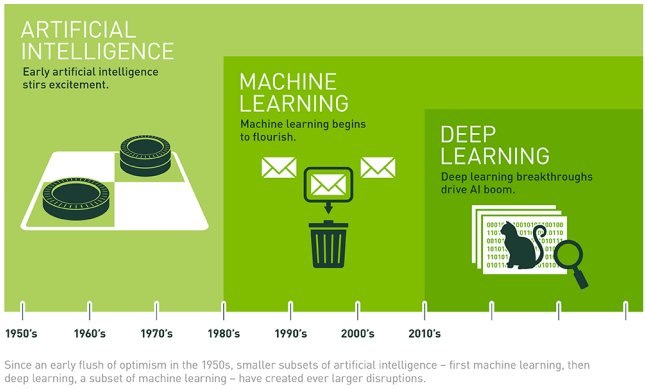
shrink over the years. All three figures 1.2 below have a new line of evolution, a breakout around the 1980s, all related to each other. Integrated circuits, personal computers, and a new concept, **Machine Learning (ML)**, which forms part of AI. In addition, around the 2010s, another new revolutionary concept, **Deep Learning (DL)**, causes AI applications boom in the market.



(a) Integrated Circuits size evolution.
Source: Computer History



(b) Smallest Computer Evolution.
Source: The Global Scientist



(c) Artificial Intelligence to Machine Learning and Deep Learning Timeline.
Source: NVIDIA

Figure 1.2: Around 1980s Artificial Intelligence starts becoming a reality.

Deep learning is a subset of machine learning and both these are subsets of Artificial Intelligence. One of the most popular machine learning application is the product recommendation or also known as targeted marketing. Netflix uses this technique to collect a user's data, "make" the machine learn from it, and classify all the different types of series or movies that the user has liked or seen previously. Amazon does the same from the previously bought or browsed products from a user's account data. Machine learning provides an output learned from a collection of data.

Deep learning, on the other hand, additionally has a variety of applications and the one this

project focuses on is the **Automated Speech Recognition** application. Starting from 2010 a whole lot of new speech processing devices and applications began to commercialize and change our daily life, letting these machines be one of our essentials.

The whole picture is not complete yet as the main fuel of artificial intelligence is still missing. Without data, the machine would have nothing to learn from. Personal computers, integrated circuits, shrinking memory cards, Internet, social media, artificial intelligence, and data are all a part of the same thing. They grow together, in parallel, and in harmony to make our life easier. Find the cycle represented in figure 1.3.

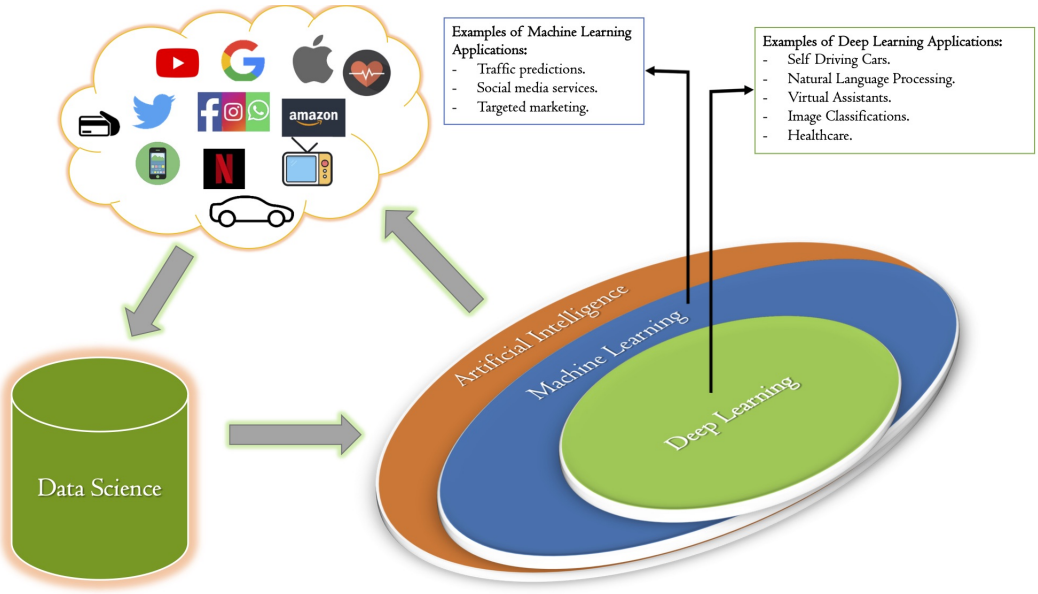


Figure 1.3: Network - Data - AI Connection

It is worth considering that the fundamental element of this cycle is **data**. Thus, for an ASR, to learn a language, a huge data-set is needed to feed the deep learning neural network. Think about a child learning to speak and understand a language, it takes several years. The child learns being retro-alimented by his/her environment repeatedly until he/she gets it; in this aspect, the neural network will also feed on a sufficient quantity of data provided, take a considerable amount of time in training, and learning our speech, reassemble the patterns and converting to text.

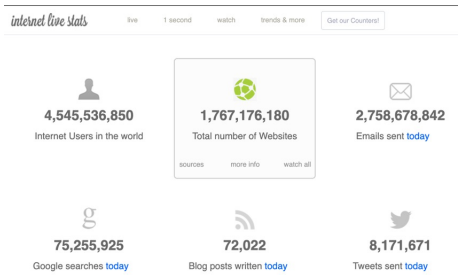


Figure 1.4: Live numbers of visits and searched per second
Source: Internet Live Stats.

The data-set handled by massive companies like Amazon, Google, Microsoft, Facebook, Instagram, etc. can be imagined by looking at the figure 1.4. Designing and building the neural network would be like Oz's tin man without a brain. Imagine a dog breeding classification problem with deep learning, not only the dog's breed has to be taken into account, their position, in a group, a same breed can come in a different color. A dog's breed is hard to even be recognizable by a human, teaching a machine to recognize it would be a tough task.

Artificial Intelligence is not being exclusively used for series or movies recommendation, it can

be and is being used in any sector. After all, AI is a machine mimicking human behaviour so it could be applied across a wide variety of industries to perform as a human or even better. In an experiment performed by Google, the new AI system performed better than radiologists in breast cancer diagnosis as reported by The New York Times [3]. The system produced a 9.4% reduction in false negatives. This study reveals that machines may be better at finding patterns than actual human eyes.

According to Gartner's Hype Cycle for Artificial Intelligence 2019, as represented in the figure 1.5 Speech Recognition is very close its mainstream adoption. The figure 1.6 shows speech recognition market's unstoppable growth, in the present and in the future.

While according to Gartner, Deep Learning is still in its early years and its future is not yet been secured. Deep learning latest picture can be found on LF AI Foundation Interactive Landscape. The interactive landscape shows 271 cards, most of them in open source code with a market capital of $12.13T and funding of $32.6B as mentioned on the website. Deep learning is a huge yes in today's market and also in tomorrow's as per the predictions.

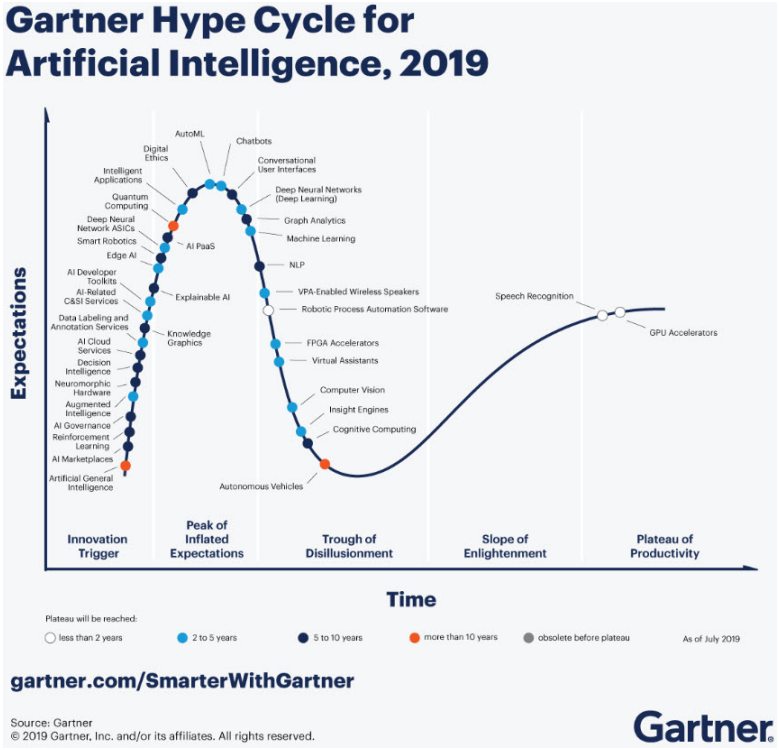


Figure 1.5: Gartner Hype Cycle for Artifical Intelligence, 2019.

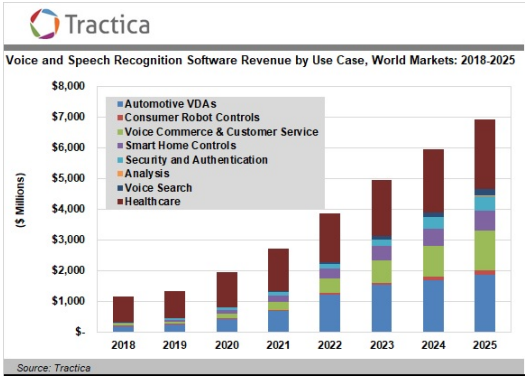


Figure 1.6: Voice and Speech Recognition Revenue by Use Case, World Markets: 2018-2025.
Source: Informa Tech

1.1 Objective

It would have been impossible to define the objective of this project without Tucuvi.

Tucuvi is a virtual healthcare assistant based on AI and voice technology. Tucuvi's main target is to guarantee continuous assistance to aged people in their own homes through landline telephones.

The actual ASR in use by Tucuvi is the **Google ASR**. The proposal of creating ASR from scratch is mainly because of economic reasons. Right now, Google's Speech to text APIs doesn't allow the models trained on telephone speech in Spanish. The cost is **0.015[Euros/min]**, which means that every minute, conversions are done per increments of $15[seconds]$.

> *"The purpose of this project (in collaboration with Tucuvi) can be divided into three main challenges:*
>
> 1. *Design an Automated Speech Recognition (ASR) using Deep Learning.*
> 2. *The ASR should be able to process in Spanish (Castellano).*
> 3. *It should be able to understand elderly people's speech."*

A high level scheme of the target solution is illustrated in the figure 1.7, the ASR to be designed is what is inside the red dotted box. In other words, the scope of this project is to train the machine to understand what the person on the other side of telephone is saying and produce text as output.

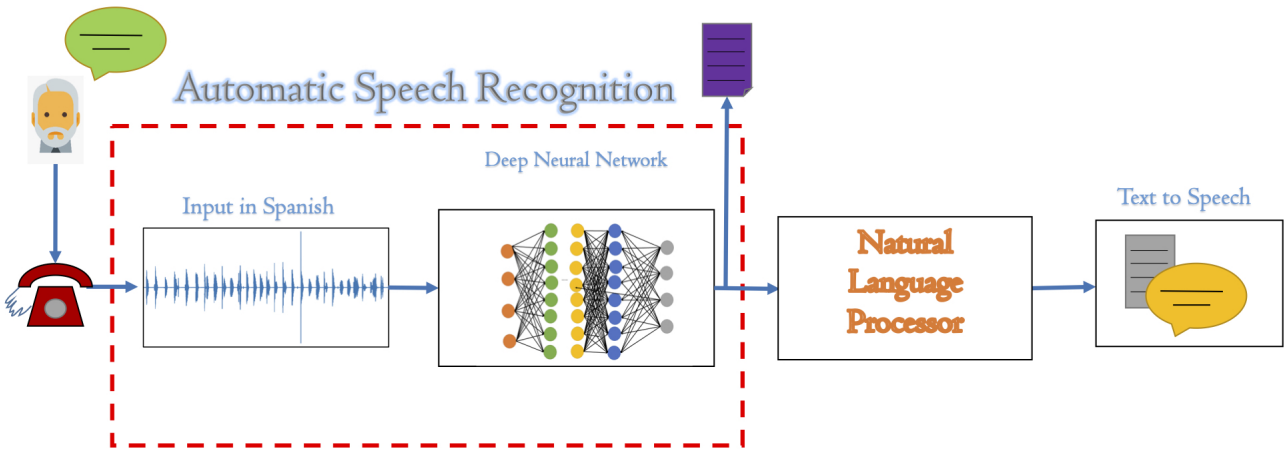


Figure 1.7: High Level Scheme of the objective described.

1.2 Social Impact

Executing this project means participating in some of the **Sustainable Development Goals**. To be exact, two of the seventeen goals, as shown in the figure 1.8

1. Good Health and Well-Being.
2. Industry, Innovation and Infrastructure.

Figure 1.8: Sustainable Development Goals
The target goals are inside the blue dotted box.

Historically, human being's lifespan has kept on increasing. In 2019, the total population of people older than 65 years old was about 20% of Spain's total population while it was only 10% in 1975, as shown in figure 1.9. Therefore, the elder population has been growing over the years. Maybe because of better healthcare throughout the years, finding the reasons is out of the scope of this project. The point worth mentioning here is that the population of elder people in Spain is a considerable one. This point entails in assisting the elderly population of our society.

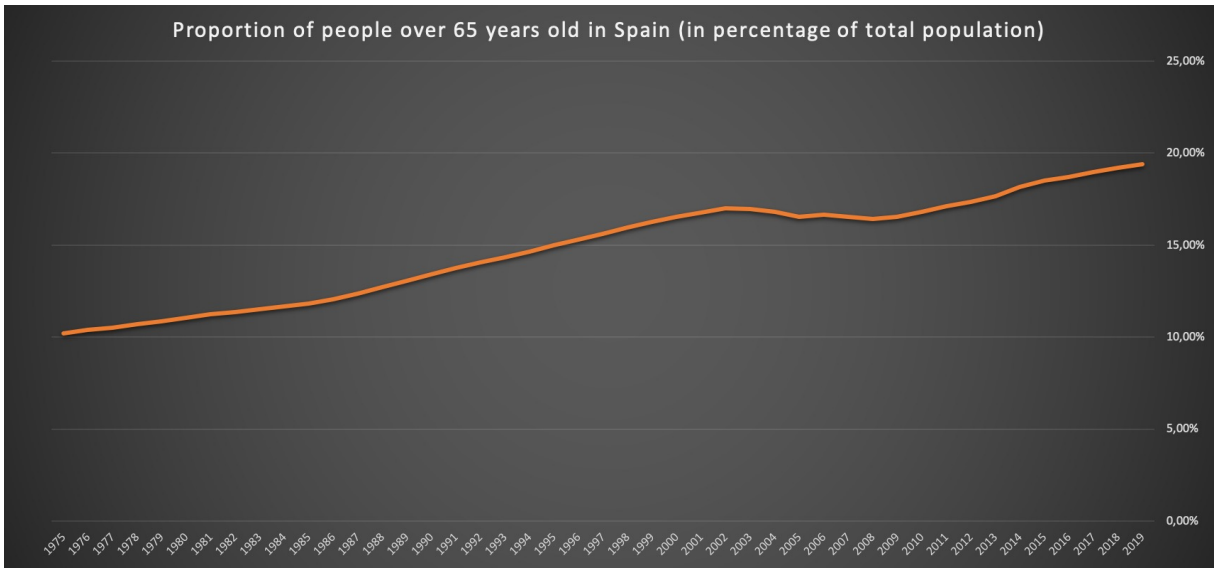


Figure 1.9: Proportion of people over 65 years old in Spain (in percentage of total population).
Source: Instituto Nacional de Estadística.

The idea is giving assistance to the elders by telephone on a daily basis call to know about their health, about how they are feeling, to kill their boredom, to give them company, to give them a friendly reminder to take their pills, etc. Nevertheless, attending almost about 20% of the total Spanish population of 47,100,396 [4] on a daily basis is over imagination as per resources.

Intelligent machines capable of having a human conversation might be the solution. In this project, having this type of conversation through any smart speaker is discarded, this type of service will be provided by a landline telephone as owning one of these is most probable than owning a smart speaker at any of their homes.

Take, for example, the recent outbreak of the virus COVID-19 (the year 2020), the elder population was the most affected. Giving them company by telephone, virtual assistance or a real one, would have been the best method to arise their hope and make them feel cared for.

1.3 Hardware and Software Resources

The following hardware and software resources were used for the development of the project:

1.3.1 Hardware Resources

- MacBook Air 2018
- Processor: 1.6GB Intel Core i5 Double Nucleo
- Operative System: macOS Cataline v10.15.4
- GPU: Explained in **chapter 4**.
- Internet Connection: Wi-Fi

1.3.2 Software Resources

1.3.2.1 Programming Languages

- **Python [v3]:** *"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.*[1]*"*
- **Pytorch [v1.5.0]:** *"PyTorch is an optimized tensor library for deep learning using GPUs and CPUs*[2]*."*

1.3.2.2 Development Environment

- **Jupyter Notebook**: *"JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data.* [3]*"*
- **Kaggle and Google Colab**: Please find the details in chapter 4.

1.3.2.3 Code Repository

- **GitHub:** GitHub has been used exclusively only as a code repository for this project, although, it also offers versioning. The code and the results in $jupyter$ notebook can be found in my repository.

[1]Definition from Python's official page
[2]Definition from Pytorch's official page
[3]Definition from Jupyter's official page.

1.3.2.4 Tools used for documentation

- **Overleaf as LaTeX editor**: *"Overleaf is an online collaborative writing and publishing tool that makes the whole process of writing, editing and publishing scientific documents much quicker and easier. Overleaf provides the convenience of an easy-to-use LaTeX editor with real-time collaboration and the fully compiled output produced automatically in the background as you type*[4]*."*
- **Microsoft Office 365**
  - **Microsoft PowerPoint**: In this project PowerPoint was used for creating diagrams. Any image without any source mentioned was created using this tool. It was used for also creating the presentation.
  - **One Drive**: In this project One Drive was used as the official tool for document storage provided by the University.

1.4 Memory Structure

This memory is divided in four chapters.

1. This **first chapter** is dedicated to a short introduction with the definition of the goal to accomplish, the possible social impact, the hardware and software resources.
2. The **second chapter** covers the explanation of the concepts used to implement the solution.
3. The **third chapter** is dedicated to **deep learning** as it is a wide subject, and it is the method used, it cannot be covered in just one section. This chapter will give the reader a much better foundation for deep learning.
4. The **fourth chapter** covers the practical side of the project in which the whole application as well as the training of the neural network will be described. The architecture, the different classes and any other subsystems used will be also explained in this chapter.
5. The **last chapter** finalizes this document with conclusions and how this project can be expanded to other applications in the future.
6. There is also a section with all the references, the visited websites, documents and bibliography used to complete this memory and the project.

[4]Definition from Overleaf's official page

2

Previous Concepts

This chapter explains the necessary concepts used to develop the solution. All the notion explained in this chapter will be helpful to the user for a better understanding. This chapter gives a brief introduction to **human speech** production and perception. This section's information has been researched from different sources but most of them were collected from [5],[6] and [7].

As the scope of this project is using deep learning, this concept will be explained in the next chapter.

2.1 Speech

Human speech production, reception, and representation had to be studied to design the system. Speech as a concept, as a communication tool, is easy to understand as this is our main communication channel. However, speech as a signal, in mathematical terms or as a tangible signal for processing is much more complex.

2.1.1 Speech Production

Speech is what human beings use to communicate ideas or thoughts that form in the brain by following any of the linguistic rules known by him/her.

This section will be dedicated to explaining on a high-level basis how human beings articulate vocal sounds and how can they be analyzed with the help of mathematics and physics.

The sound is produced by the vocal tract. The vocal tract's anatomy is represented in figure 2.1.

The lungs produce air and create air pressure. This air pressure passes through the vocal cords and thus creating vibrations or waves or oscillation. These vibrations once reach the articulatory organs (pharynx, tongue, vocal/nasal cavity, etc.) it can be modified to articulate sound/phoneme depending on the movement of the jaw, tongue, etc.

The vocal tract transforms raw sound produced in the vocal folds into recognizable sounds.

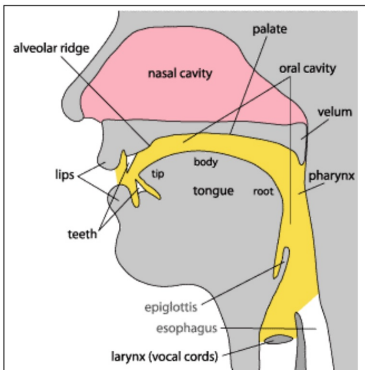


Figure 2.1: The human vocal tract.
Source: Facultad de Filología de la Universidad de Sevilla.

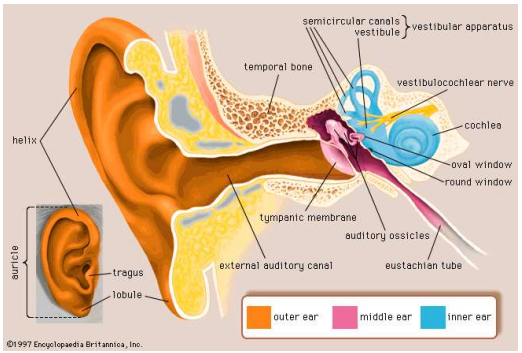


Figure 2.2: The human auditory system.
Source: ENCYCLOPÆDIA BRITANNICA

2.1.2 Speech Perception

This section is the most interesting one for designing ASR as it will cover the process of human speech perception. Following this approach will help create better designs, robust and efficient systems that approximate how the human brain processes the sound waves.

The human auditory system consists of the outer, middle, and inner ear as shown in figure 2.2. The functionality of each of the part of the auditory system are described below:

- The auricle (also known as pinna) collects the acoustic waves (variations in air pressure) and carries them through the external auditory canal. The auditory canal amplifies this wave.
- The middle ear acts as a transducer and converts the acoustic energy to mechanical energy. One of the functions of the middle ear is *"cushioning"* the excessively intense sounds.
- The inner ear, full of liquid, is responsible for transforming these mechanical oscillations into nerve impulses and send them to the brain through the cochlear nerve.

In summary, the human ear's function is to convert the auditory signal or sound waves to neural signals. This is called **sound perception**. The human ear limitation for sound perception is from **20[Hz] to 20[KHz]**. This means that the human ear can convert only the sound waves traveling within these frequencies.

There is a reason for humans owning two ears. With **two** ears human have the ability to:

- **Sound localization:** Even without seeing the source of a sound, human ears can localize the origin of the sound. This can be related to stereo, mono, Dolby digital sound technologies.
- **Sound cancellation:** Human is capable of obviating the sounds, noise, or unwanted voice to focus on what matters at that moment. A good example is talking with a friend in a crowded environment like a bar or a restaurant.
- Similar effects of hearing when using a headphone: The sensation of perceiving the sound inside the head.

A brief introduction to the physical dimension of sound was given in the previous section 2.1.3. The auditory perception is a branch of **psycho-physics**, a relationship between perception and physical properties of stimuli.

The **perceptual dimension** is a human mental or neural experience, whereas the **physical dimension** is an approximated measurement of the perceptual experience. In table 2.1 the equivalents between these two dimensions can be found.

Perceptual Dimension	Physical Dimension
Pitch	Frequency
Loudness	Intensity
Timbre	Spectrum Envelope or Amplitude Envelope

Table 2.1: Perceptual and Physical Dimensions.

The three physical dimensions will be discussed further more in the following section with more details.

2.1.3 Physical Dimensions of Sound

Few concepts need to be revised before diving into the three dimensions mentioned in the previous section.

2.1.3.1 Sine wave

- A **sine wave** is the purest and the simplest form of sound which is not found in nature. It is also called **pure tone** and it will be taken as reference in this project as speech is nothing but a complex combination of different types of sine waves.
- An **amplitude** is the maximum height reached starting from axis 0.
- When the sine completes 180º and makes a complete cycle, it is called a full cycle or **period** and is noted as **T**.
- **Frequency** is the number of complete cycles that fit in one second. It is measured as the inverse of the period, $f = \frac{1}{T}$, it is measured in Hertz [Hz] or s^{-1}.
- The distance covered by the sine wave in one period is called **wavelength**. It is noted as λ. It is inversely proportional to frequency, hence, the longer the wavelength, the lower is the frequency.

All the definitions explained are represented in the below figure 2.3. The sound wave can be expressed as a sine wave as shown in the equation 2.1, where A is the maximum amplitude, f is the frequency of the wave, ϕ is the phase, x is time and y is the amplitude of the sound at time x.

$$y = A\sin(2\pi f x + \phi) \tag{2.1}$$

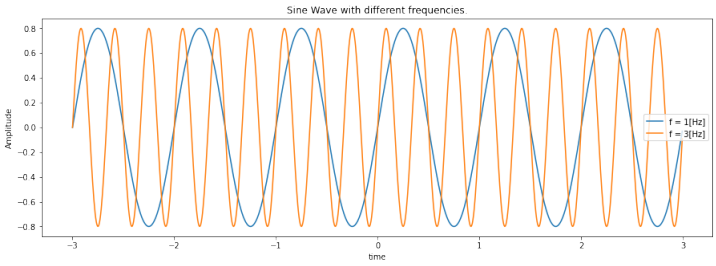
2.1.3.2 Loudness

As stated in table 2.1, loudness is a subjective measure of perceived sound intensity. The *sensation* of loudness is related to sound pressure allowing to distinguish between soft and loud sounds.

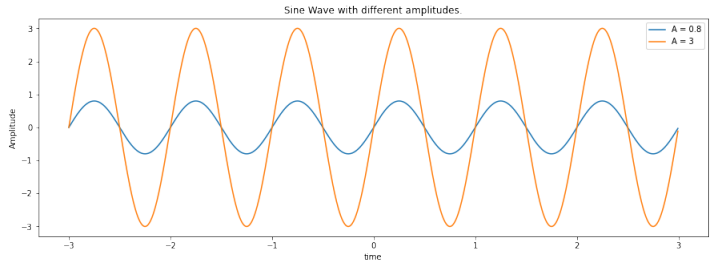
The sound intensity is expressed in Watt by square meter $\frac{W}{m^2}$. Loudness is related to distance as power decreases as the distance increases and vice versa. Loudness is measured in decibels.

The **threshold of hearing (TOH)** is the amount of sound pressure required to create a sensation in human ear. The threshold of human hearing is $I_{ref} = 10^{-12}[\frac{W}{m^2}]$. And the **threshold of pain** is $I_{pain} = 10^{1}[\frac{W}{m^2}]$ which is 10^{13} times the threshold of hearing. The range of intensities that a human ear can hear is very large, therefore, **decibels** (power of ten) is the scale used to measure sound intensity and its relation with intensity is as shown in the equation 2.2

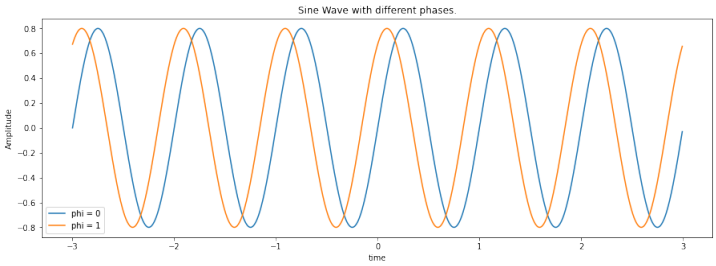
$$dB = 10\log\frac{I}{I_{ref}} \tag{2.2}$$



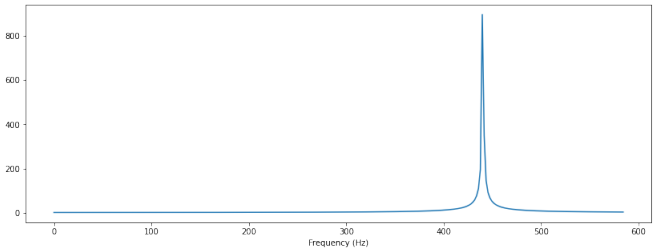
(a) Sine wave with different frequencies.



(b) Sine wave with different amplitudes.



(c) Sine wave with different phases.



(d) Pure tone on 440[Hz] in frequency domain.

Figure 2.3: Different Representations of the Sine Wave.

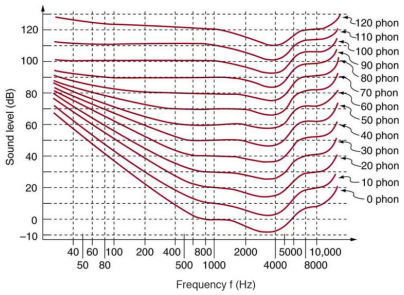
The figure 2.4a is called the Equal Loudness Curves, and it represents the relationship between the loudness, intensity levels for different frequencies (these apply to healthy human ears). **Phon** is a unit used to measure loudness in numbers. Phons and decibels measure the same at f = 1000[Hz].

2.1.3.3 Pitch

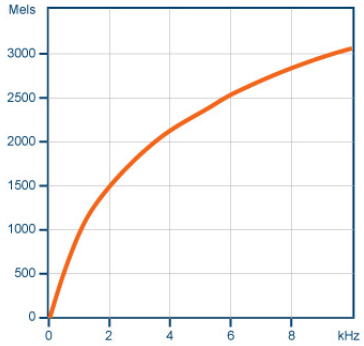
Pitch is a physiological parameter that is related to the perceived frequency by the human's auditory system. At higher frequencies, the pitch is perceived as high tone (not the same as loudness) and at lower frequencies, the pitch is perceived as low tone.

Mel-scale is used for measuring pitch concerning the frequency. It is a logarithmic scale and is the one which approximates the most to how human perceive a frequency.

Analyzing the figure 2.4b, the perception of pitch is not linear and the Pitch (mels) can be



(a) Equal Loudness Curves
Source: Lumen Learning Courses, chapter 17.



(b) The Mel Scale
Source: Cochlea EU

Figure 2.4: Loudness and Pitch Scales.

calculated as shown in the equation 2.3.

$$Pitch(mels) = 1127log_e(1 + \frac{f}{700}) \tag{2.3}$$

As a curiosity, Mel2Hz calculates very easily the conversion from frequency to pitch (how frequency is perceived by human).

2.1.3.4 Timbre

Timbre is a subjective term, the same loudness and pitch can be perceived differently by humans. For example, the same note played by a guitar and a flute doesn't produce the same timbre. It is the quality of sound.

2.1.4 Sound Representation

A voice is a complex sound. Each phoneme has a characteristic spectrum of frequencies, and everyone's voice differs regarding pitch and timbre. There are different forms of representing speech, in this section, a demonstration will be shown in time, frequency, and spectrogram domain.

For a better explanation, two figures 2.5 and 2.6 were plotted using the $jupyter$ notebook $audio\_figures.ipynb$. The data samples used are from the Google library, speech commands [8].

To create the figure 2.5, two audio data were taken produced by two different female voices saying the word *happy*. For comparison, the other two audio data were taken produced by two different male voices saying the same word.

To create the figure 2.6, two audio data were taken produced by a man and a female for the word *three* and the word *yes*.

2.1.4.1 Time Representation

As seen in figure 2.5 and 2.6, the wave plot is a complex signal, a combination of different waves represented in the time domain. As an observation, even the same word *happy* have different waveforms. This probably can be the cause of background noise or even the difference between male and female voices. Also, all four voices are from different four people.

2.1.4.2 Frequency-Amplitude Representation

All the amplitudes are crowded in the lower frequencies as shown in both figures 2.5 and 2.6. This representation doesn't give us any further information to differentiate between different types of words as *happy* on the figure 2.5, *three* and *yes* on the figure 2.6.

2.1.4.3 The Spectrogram

The spectrogram is a three dimensional representation as shown in both figures 2.5 and 2.6. It represents the sound's intensity (in DB) for each frequency over time. If both figures are compared, there seems to be a pattern more visible in the spectrogram. This is the representation that is most used for speech processing.

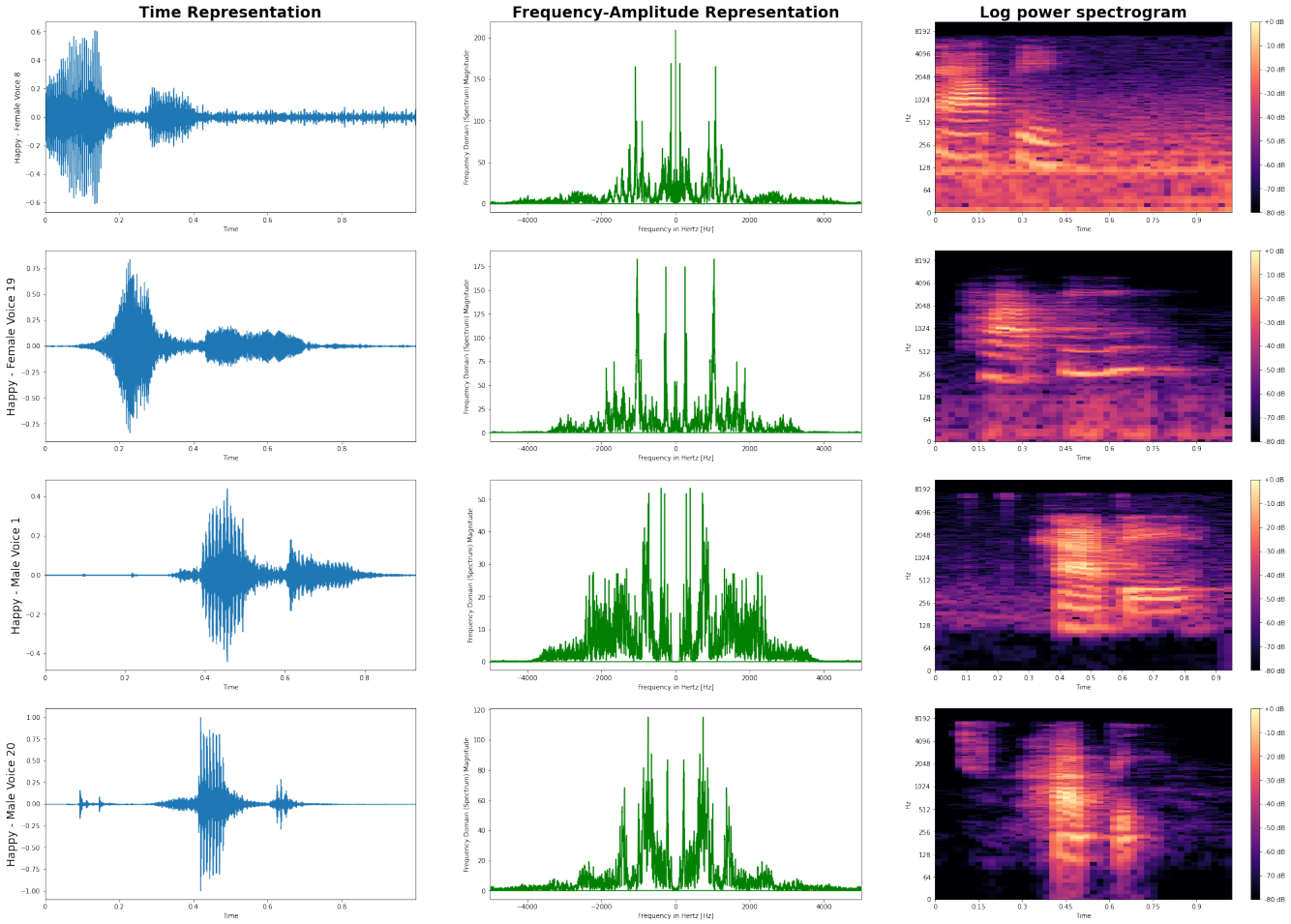


Figure 2.5: Three different Sound representations of the word *happy* spoken by both female and male.

When the signal is represented in only time, the frequency information is lost. When the signal is represented in only frequency, there is no means of knowing the time of the frequency. The spectrogram is widely used in speech processing because it provides information about speech in both time and frequency. The frequency-domain in figure 2.7b provides information about frequencies but not about time. The spectrogram in the same figure's information extends to frequency, domain, and amplitude of the same word.

In linguistic, *reading* spectrograms is a general method of understanding language and phonetics.

The frequencies depend on how words are articulated. **Formants** represent the range of fre-

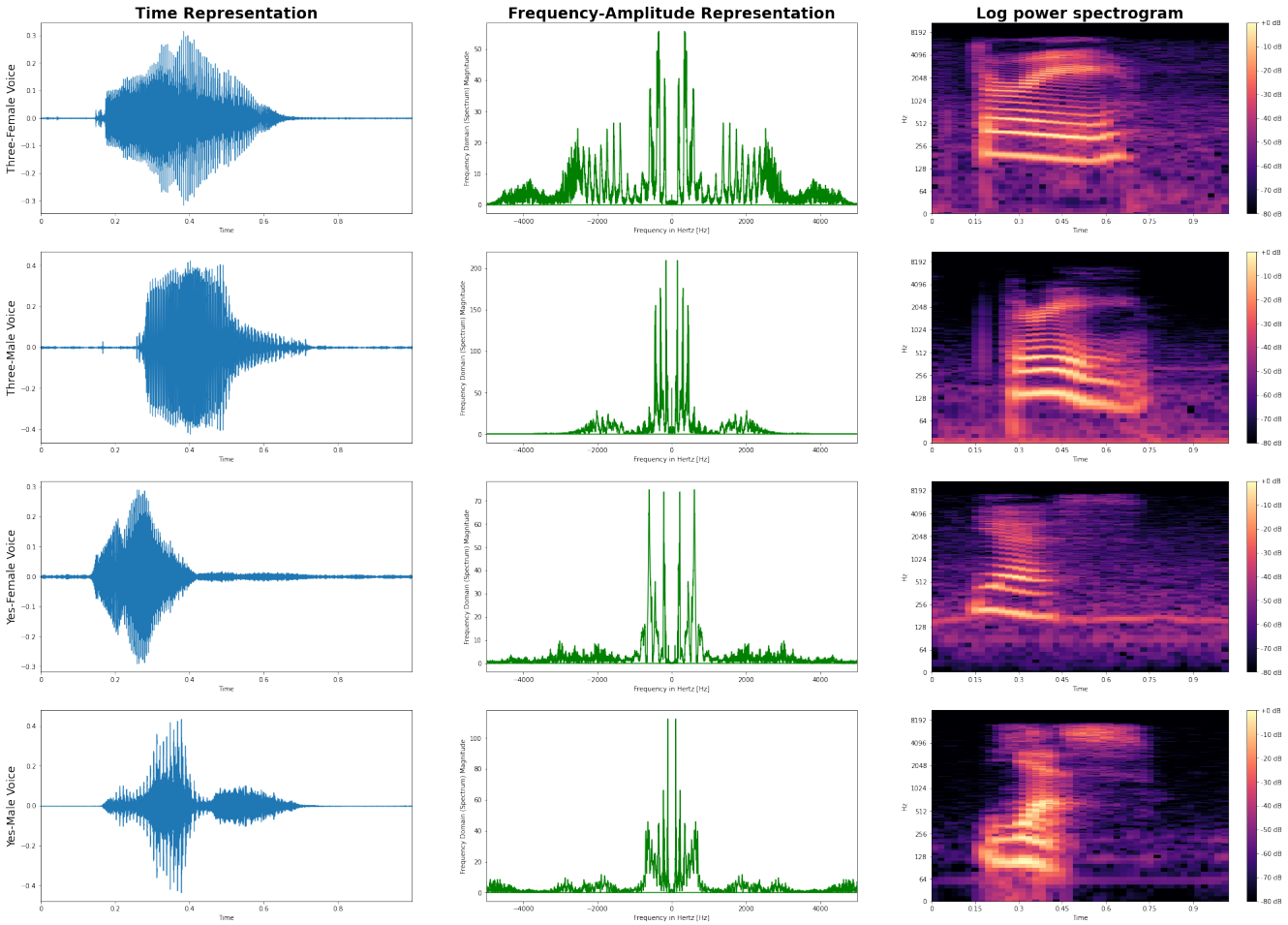
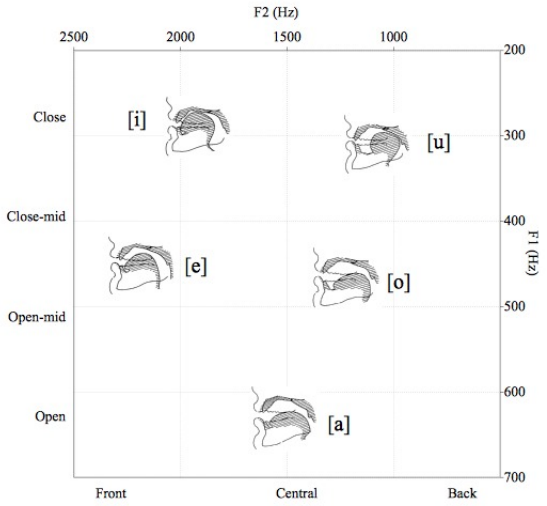


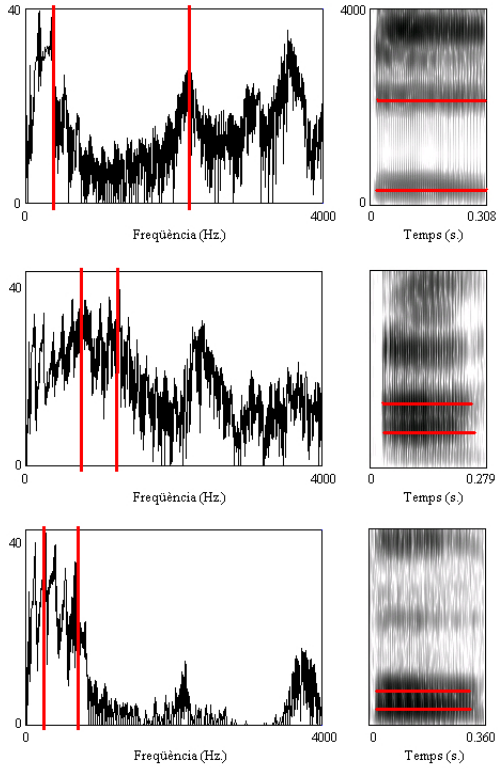
Figure 2.6: Three different Sound representations of the word *three* and the word *yes* spoken by both female and male.

quencies that are amplified and cause peaks of intensity in the spectrum domain. Figure 2.7a represents the relationship between the vocal articulation and the formants when the vowels are pronounced in Spanish.

The calculation of spectrograms is explained in the next section.

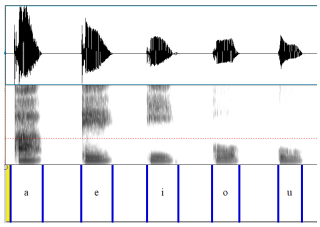


(a) Vocal articulation and Vowels Relation in Spanish
Source: Grup de Fonètica. Universitat Autònoma de Barcelona

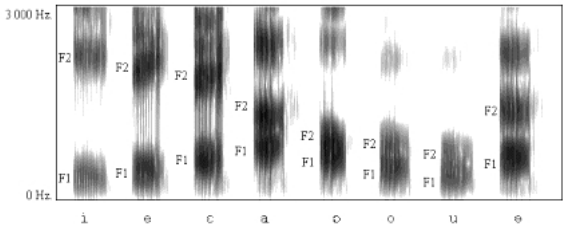


(b) Frequency domain vs. Spectrogram
Source: Grup de Fonètica. Universitat Autònoma de Barcelona

Figure 2.7: Reading Spectrograms in Spanish.



(a) Waveforms & Spectrograms of Spanish vocals.
Source: Recorded and analysed with Praat.



(b) Spectrograms of Catalan Vowels.
Source: Grup de Fonètica. Universitat Autònoma de Barcelona

Figure 2.8: Spanish and Catalan Vowels' Spectrograms.

2.1.5 Signal Processing

The previous section covered the sound's physical dimensions and different types of representation. This section will explain more about the signal pre-processing needed before feeding it to the Neural Network. The pre-processing of the signal can be understood as converting the audio signal to "another language" that the machine can understand and find patterns.

2.1.5.1 Sampling

Any audio signal like most of the real-life elements depends on time, and time is continuous but computers only understand digital signals. Thus, a pre-processing is needed to convert these signals to digital ones.

Sampling is the method that is used to convert the continuous signal to a digital one. It is referenced as F_s. The technique consists of taking a number N samples per second [9]

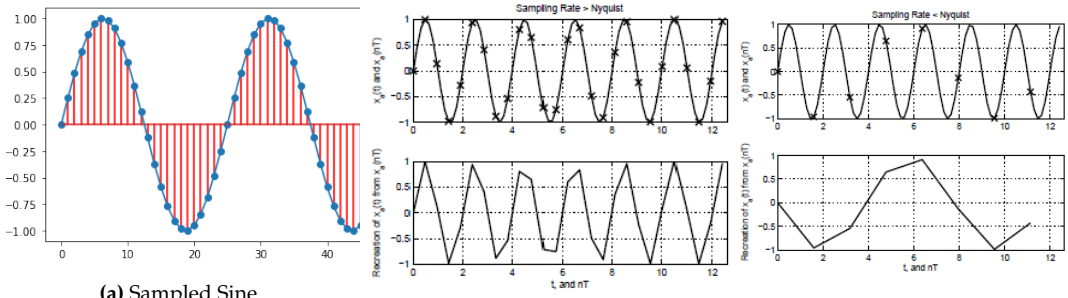
The Nyquist theorem should be applied to guarantee the reconstruction of the original signal. If the maximum frequency of the signal is F_m, then the sampling rate should be at least two times the maximum frequency of the original waveform 2.4.

In the figure 2.9a, the F_s is 50 samples per second.This means that every T_s 20[ms], the signal's amplitud is measured and saved.

When the sampling frequency is greater than twice the Nyquist frequency 2.4, the signal can be reconstructed as shown in 2.9b.

When the sampling frequency is less than twice the Nyquist frequency 2.4, the original information is lost and the signal cannot be reconstructed as shown in 2.9c.

$$F_s \geq F_m \tag{2.4}$$



(a) Sampled Sine
Source: From my notebook
$audio\_figures.ipynb$

(b) Correct Sampling
Source: [10]

(c) Aliasing Effect while sampling
Source: [10]

Figure 2.9: Examples of sampled signals.

2.1.5.2 Spectrogram Calculation

The **Fourier Transformation**, also known as FT, is a mathematical calculation to convert time into frequency domain. Fourier's theory proves that any sign can be represented as a sum of cosine and sines of different frequencies.

The equation 2.5 is used to convert time to frequency for analog or continuous signals and the equation 2.6 is used for discrete or digital signals.

$$\mathscr{F} \longrightarrow X(w) = \int_{-\infty}^{\infty} f(x)e^{-jwt}dt \tag{2.5}$$

$$\mathscr{F} \longrightarrow X[n] = \sum_{n=-\infty}^{\infty} x[n]e^{-jwn} \tag{2.6}$$

However, the **Fourier Transform** can be applied only on infinite periodic signals so, in practice for speech signals, the method used to convert them to frequency-domain is called **Short Time Fourier Transform**, mostly known as STFT.

The method consists of taking a small window of the signal and calculating the FT only for that short period of time. These windows are then allocated successively in the time domain with their respective frequencies [9].

Below the equations 2.7 and 2.8 are for continuous and discrete signals respectively. x(t) or x[n] are both multiplied by the function w which represents the window function used to extract the signal for a determined period of time.

The window function w is slided to the right as m increases and thus resulting $x[n]w[n-m]$ or $x(t)w(t-\tau)$ and then the FT is applied to these resulting signals.

$$\textbf{STFT}\{x(t)\} = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-jwt}dt \tag{2.7}$$

$$\textbf{STFT}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-jwn} \tag{2.8}$$

The outcome of the STFT then needs to go through one last step so that we can also be informed about the phase and not only the energy. As already explained in previous sections, the scale that approximates to how human perceives speech is the logarithmic scale. So, finally the spectrogram can be calculated as shown in the 2.2

$$log|X[k]|^2 = log({X_r}^2[k] + {X_t}^2[k]) \tag{2.9}$$

The figure 2.10a illustrates the spectrogram calculation in a screenshot. Overlapped triangle function g[n] is used to not loose context of the signal. DFT is acronym of Discrete Transform Fourier as described in the equation 2.6.

Henceforth, all the signals will be exclusively mentioned in its discrete form as digital signals will be processed.

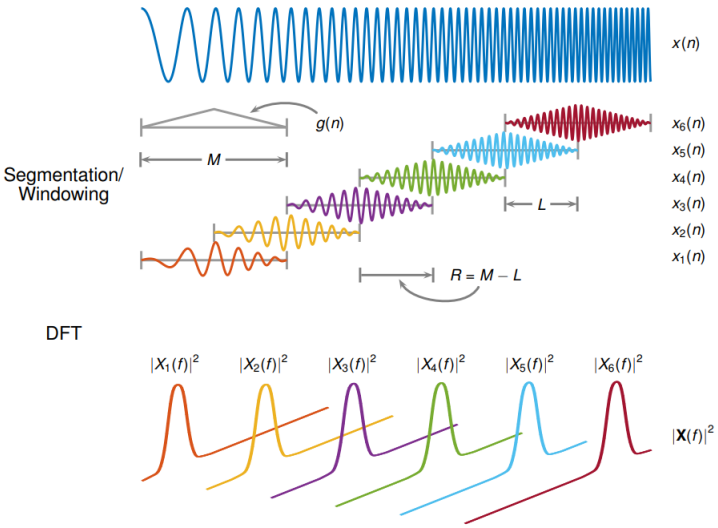
For this project, the python library **librosa** was used to calculate these spectrograms. Librosa uses as per default a sample rate of 22050. Both figures 2.6 and 2.5 were created using the mentioned library. Anyhow, the gray spectrograms differ from those like happy and yes ones. Actually, the calculation of both is different.

Already mentioned in the section 2.1.3.3, the **mel-scale** is the most approximate scale to the one of a human's ear. How to achieve this scale computationally? This is achieved by using triangular filter banks called **Mel Filter Banks**. Same as human ear, these filterbanks are non-linear, these filter concentrate more on lower frequencies (similar to human ears) and less in high frequency regions as shown in figure 2.10b. In the m_p, p is the number of mel bands, meaning the product of this filter with the spectrogram will provide m_p mel bands, commonly known as features.

After obtaining the power spectrum as shown in figure 2.10a, this is rotated by 90º as shown in the figure 2.10d and the amplitude is mapped to a grey scale (0-255), wher the darker areas are repsented by higher amplitudes. Now, this is what is represented in figure 2.8.

The process is not complete yet because human ear is non-linear, this approximation can be calculated by applying the **Mel-Frequency Filter**. After applying this filter bank to each vector, what is obtained are m_p features of the audio in t time and this known as the **Mel-Spectrogram**. The complete picture as shown in figure 2.10c.

Spectrograms are heavily used in speech recognition systems. The most known ASR is **Deep Speech**, it is an open source library and can be tested in Mozilla's research website. Further details will be discussed in chapter 4.



(a) Spectrogram Calculation in a screenshot
Source: Matlab

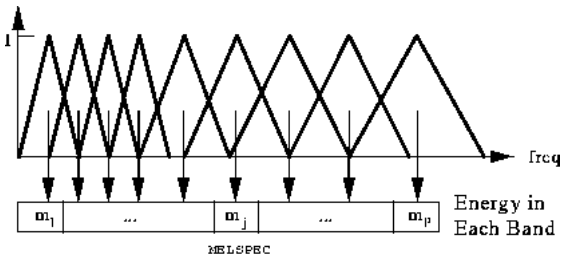
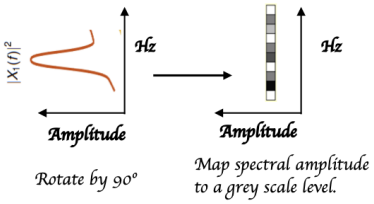
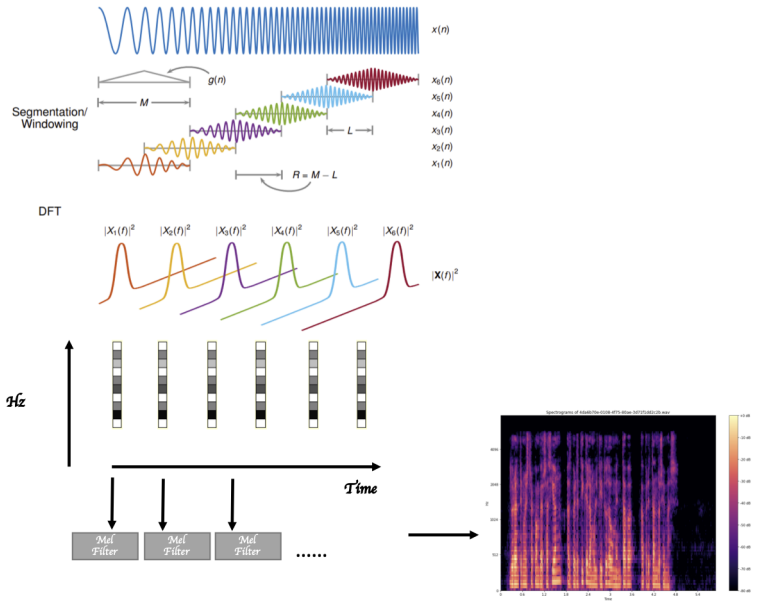


Fig. 5.3 Mel-Scale Filter Bank

(b) Mel-Scale Filter Bank.
Source: The Laboratory for the Recognition and Organization of Speech and Audio



(c) Rotate and map to grey scale.
Source:



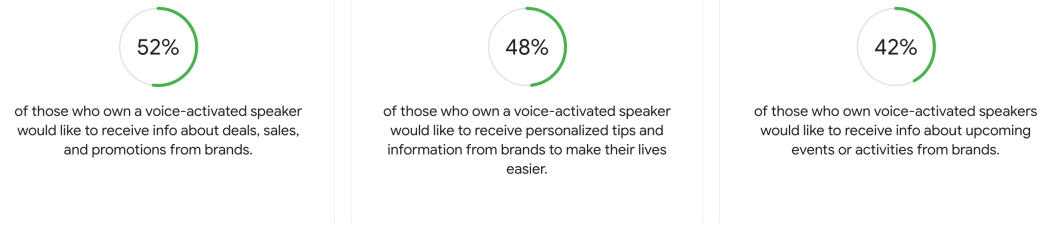
(d) Complete picture to get the Mel Spectrogram.
Source: Falta

Figure 2.10: Mel Spectrogram Calculation Step-by-Step.

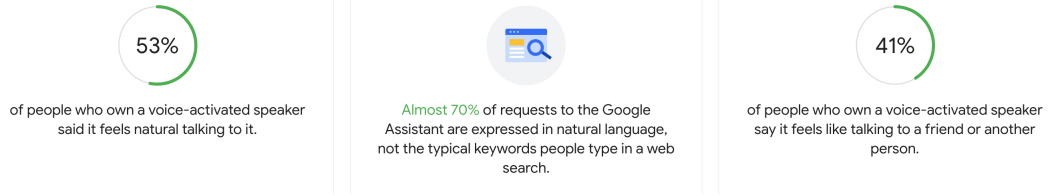
2.2 Automatic Speech Recognition

Also known as ASR or also as **Speech-To-Text**. It is an application that turns speech to text and that text can be processed to take action. For example, Alexa converts speech commands to text, takes action, and produces an output.

Voice command applications are increasing day by day and it is impacting our lives in many ways. Some examples are shown in figure 2.11. According to Google Insights, ASR is impacting customer behaviors.



(a) ASR impacts people's shopping
Source: Google Insights



(b) People are positive about ASR.
Source: Google Insights

Figure 2.11: People's opinion on ASR from Google Insights.

2.2.1 History

Speech Recognition's evolution can be divided in two eras, before and after AI. The table 2.2 lists the main events in speech recognition's history before AI.

The modern era of speech recognition can be found in the table 2.3

Table 2.2: Speech Recognition's Early Days Before AI

1950's	Bell Labs designed *"Audrey"*[11] intending to load the burden of the secretaries while dictating. Audrey could recognize 9 digits spoken by a single voice.
1960's	IBM's *"Shoebox"*[11] is able to differentiate between 16 words. It is also able to do simple maths calculations via voice commands.
1970's	The U.S Defense department understands the importance of speech recognition and funds (for five years) the DARPA SUR (Defense Advanced Research Projects Agency Speech Understanding Research). Harpy is the system that comes out of this research and is able to recognize a minimum of 1,000 words.[12][13]
1980's	Researchers start using **Hidden Markov Model (HMM)** [14] for speech recognition, one of the most important breakthrough in speech recognition. HMM is an algorithm to determine probability of a word through patterns from the unknown or "hidden" parameters.
1980's	IBM's *Tangora*[11] recognizes 20,000 spoken words. Tangora's aim was to become a voice-activated typewriter. It recognized words and typed them on paper. Each individual had to train the system with their voice and had to pause between words.
1990's	Dragon (now acquired by Nuance) comes out with Dragon Dictate. Originally developed by Dragon for Windows. It was expensive, about 9000$.
1992	The Sphinx-II Speech Recognition System was the first one that was speaker-independent. The authors confirm that the larger the training data, model's erros decreases.[15]
1993	Apple develops a protoype called Casper for their computers[16].
2005	Google's first approach of creating a speech recognition by creating GOOG-411[17]. Telephone-based speech recognition system for directory search. This data was later used to improve their recognition systems (Google Voice Search) available in smartphones (reachable to millions and millions of people).

Table 2.3: Speech Recognition with AI

2000's	HMM is stil used but combined with NN [18].
2006	Alex Graves and Navdeep Jaitly publishes Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks[19] in their paper that the combination of CTC and RNN is a lot better than the hybrid HMM and RNN. This was the first attempt of an end-to-end speech recognition. It doesn't work as expected and the system makes spelling mistakes.
2011	Apple launches *Siri* which was similar to Google's Voice Search.
2013	Microsoft introduces *Cortana*[20][20].
2014	Amazon launches *Alexa*, available only for prime members.
2015	Baidu introduces a better version of Deep Speech called *Deep Speech 2: End-to-End Speech Recognition in English and Mandarin* and is tested against English and Mandarin successfully. It now combines CNN, RNN, and fully connected layers. It reports 7x speedup over the previous system and also that it can reduce the error rate in English up to 40%. It also recognizes Mandarin words.
2016	Google launches *Google Home*[20].

3

Deep Learning

Deep comes from layers and layers of transformations of the input data to create the desired input. As seen in the first chapter, DL is a class of ML algorithm or even an evolved branch of machine learning. In this section, questions like what is deep learning, types of deep learning, artificial neural networks, training, and everything related to deep learning will be explained. This chapter begins with a high-level foundation of artificial intelligence and machine learning to a detailed explanation of deep learning.

3.1 Artificial Intelligence

Any action mimicking human behavior by a machine can be described as artificial intelligence. AI is computer science's branch: *the effort to automate intellectual tasks normally performed by humans* [21].

acrshortai can be divided into two categories. The first one is comparing the machine's intelligence to a human being or the intelligence gained by the machine and the second one is based on functionalities.

Compared to human's mind

1. **Artificial Narrow Intelligence (ANI)**: Applications programmed to perform one specific narrow task autonomously. For example, a chess machine, chatbots, Siri, OK Google, or Alexa. Speech Recognition falls in this category because it is designed to realize one limited task, which is to interpret human speech and convert it to text.

2. **Artificial General Intelligence (AGI)**: These applications or robots are comparable to human's intelligence. The AI under this category has the same level of perception, understanding, and complexity of a human mind. However, it is still an emerging field. The most advanced humanoid is called Sophia and was declared the first robot citizen of Saudi Arabia [22] Although Sofia has a human face and expressions, her co-creator, Ben Goertzel, confessed that she cannot be considered as AGI, she is rather a chatbot with a face.

   If I show them a beautiful smiling robot face, then they get the feeling that AGI may indeed be nearby. - Ben Goertzel [23]

3. **Artificial Super Intelligence (ASI)**: Super intelligent agents or robots would not only be machines

but it is more linked to emotional relationships, feelings, the human's emotional domain which separates a machine from a human.

Per functionality

1. **Reactive Machines**: Systems that create an output using an algorithm from inputs but it can't learn. The most famous AI system of this category is the IBM Deep Blue, a chess machine who won a human. It doesn't require recurring to a past memory or storing any to make the next move. Its stimulus is limited to the algorithm, the chessboard, and the chess pieces.
2. **Limited Memory**: In present days, AI systems of this category can be found in abundance. For instance, speech recognition is one of them, it accumulates data and learns from that data to realize a specific task.

   A self-driving car learns from data previously learned rules like lanes, safety distance combining with present data like approximating vehicles, curves, speed.
3. **Theory of Mind**: The two categories from now on remain as a theory for now. Theory of Mind [24] is a psychology term, the AI should be able to understand the humans surrounding it, as in beliefs, emotions, thoughts and adjust their behavior accordingly.
4. **Self Awareness**:

   "When attention is directed inward and the individual's consciousness is focused on himself, he is the object of his own consciousness—hence 'objective' self awareness" - Duval & Wicklund [25]

   What if robots start being self-aware of themselves and lead the human race to extinction?

Articles from [26], [27], and [28] were thoroughly read and validated to recompile information about AI categories

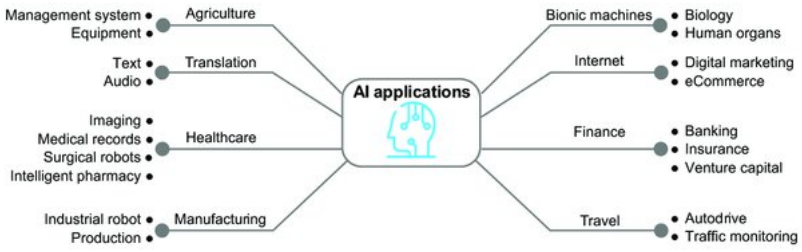


Figure 3.1: AI Applications per Industry
Source: Advancing Drug Discovery via Artificial Intelligence [29].

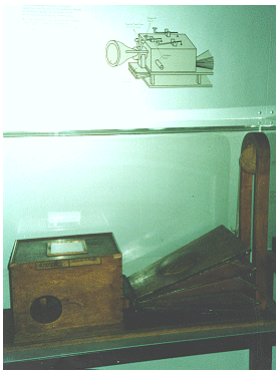
First AI application related to speech

The first AI system related to speech was **Kempelen's speaking machine** created by **Wolfgang von Kempelen** and was explained in his book *Mechanismus der menschlichen Sprache nebst Beschreibung einer sprechenden Maschine (1791)*. It is still preserved and can be found in the Deutsches Museum in Munich [30] as shown in the figure 3.2a.

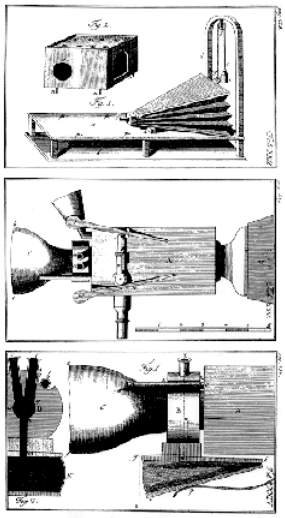
The machine can form words and a few sentences in Latin, Italian, and French. It has lungs, mouth, nose, and vocal cords as shown in figure 3.2b. Manual manipulation is needed to make it "speak" or pronounce any word and has to be played like a musical instrument.

A virtual visit and a thorough explanation of the machine can be found in Google Arts and Culture.

Moving back to present days, as mentioned in the first chapter 1 AI has a narrow relationship with big data. The present landscape of big data and AI can be seen in figure 3.3.



(a) Kempelen's Speaking Machine
Source: Stockholms Universitet



(b) Kempelen's Speaking Machine Original Sketch
Source: Stockholms Universitet

Uncountable and unimaginable applications can be thought of as AI, it is present in our daily life, in our cars, applications we use like Siri, OK Google, Google Assistant, chatbots, healthcare applications and a large list of etcs. It has become part of our life because they provide us with comfort and make our daily tasks easier.

Before closing this section, it is worth mentioning that ethical questions arise in the AI matter. If a being is created on a human's image, then shouldn't it be equal to humans? Sophia, when given the citizenship, concerns about voting, marrying arose. If Sophia is **switched off**, would it be killing her or nothing more than disconnecting any other electronic device?

In chapter 1, the relation between AI, ML, and DL was briefly explained. In the following sections, DL will be explained in detail and ML will be introduced to have a context and to compare it with DL.

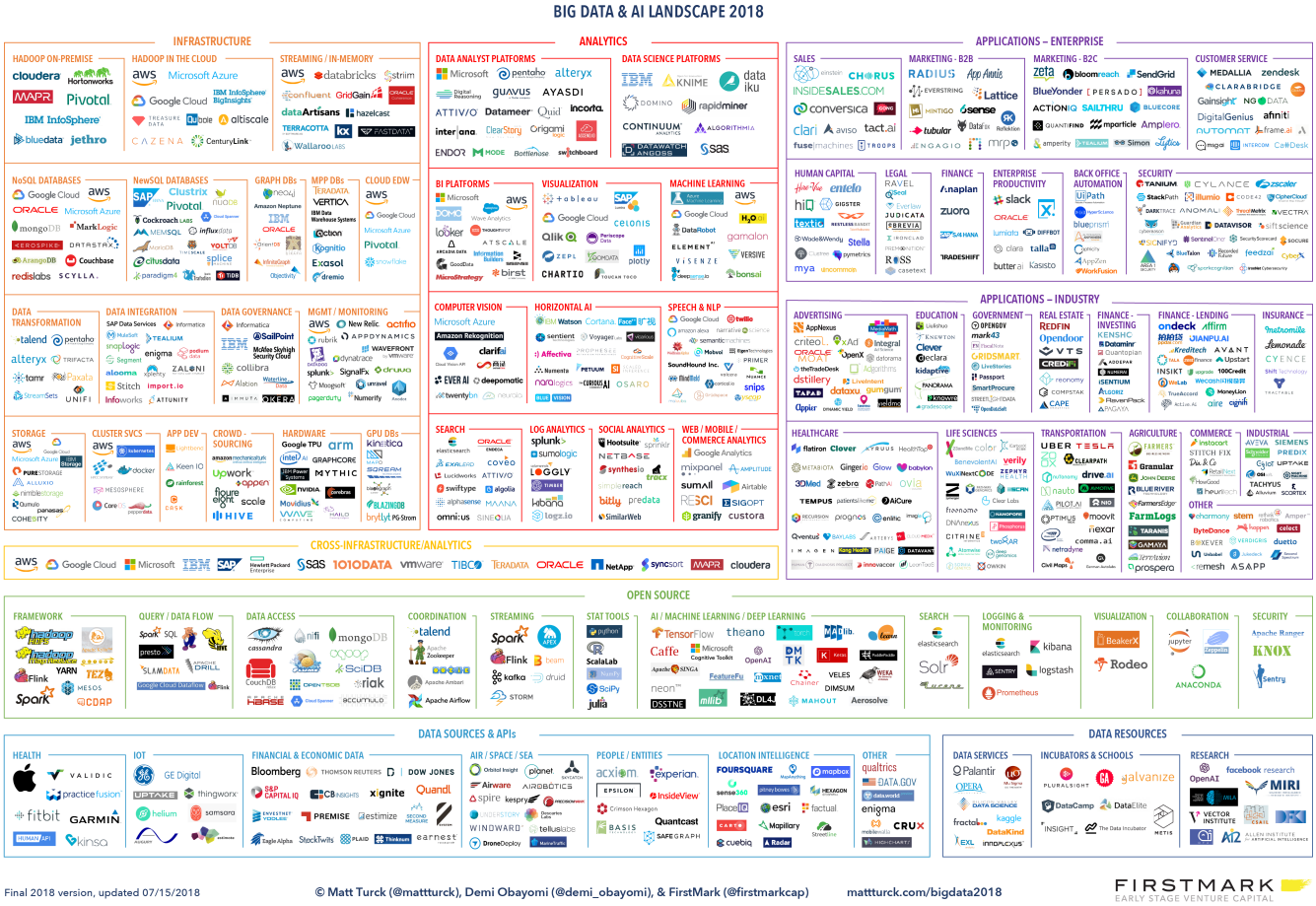


Figure 3.3: Big Data and AI landscape 2018
Better Resolution can be found here.

3.2 Machine Learning

In the previous section, AI was introduced as a branch of computer science. Machine Learning is a subset of AI. Systems in ML are sufficiently autonomous to learn from data and produce the desired outcome.

Machine Learning is a topic that can be discussed in extensive as it is a massive branch. Only few will be covered as an introduction to ML. Deep Learning is quiet a small subset of machine learning as ML uses several algorithms.

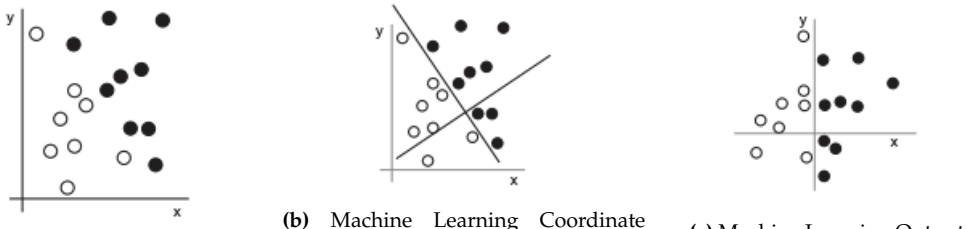
- **Supervised Learning**: The system learns by comparing its output (predicted output) with the labelled data provided a the beginning of the process. Classification and regression are two techniques in this category. Image classification is a good example and for regression predicting someone's salary given his age.
- **Unsupervised Learning**: The agent exhaustively explores the data and finds patterns. The data provided to the agent isn't labelled. Clustering is the most common technique in this category. Applications like clustering fake/real news, spam/good emails, etc. are good examples.
- **Semi-supervised Machine Learning**: The agent is provided with both labelled and unlabelled data.
- **Reinforcement Learning**: It is based on giving positive signals when the agent's learning is correct and negative when incorrect.

As an example, a classification problem can be classifying the white dots from the black dots as shown in figure. Applying machine learning to solve this classification problem means to create a **model** which takes the dots as input and creates an output which is a better representation of the classification.

In the example, *learning* means to constantly check the output produced, compare the classified output with the correct ones (feedback) and find the best representation to classify the black and white dots.

Model means what makes the transformation of the input to the desired output. For example, in speech recognition, by designing and applying a model, speech will be converted to text which is the desired output. Until now, only high-level details have been seen. From the next section, deep learning will be explained in detail, always relating to the word done in this project.

This section was written after reading articles [31], [32], [33], and the book [34].



(a) Machine Learning Input Data

(b) Machine Learning Coordinate Change

(c) Machine Learning Output

Figure 3.4: An example of classification problem.
Source: Deep Learning with Python [21]

3.3 Artificial Neural Networks History

Neural Networks have been lurking from the past sixty years, although it has recently started becoming one of the favorites in AI world. To better understand the concept of ANN, a brief history has been described below.

A Brief History of Neural Networks.

Artificial Neural Networks, ANN, were inspired by the human brain, the perfect learning machine. In figure 3.5a the dendrites receive inputs or signals from other neurons, creates an output, and sends it through the axon to other neurons. A **perceptron** is the main block or the *neuron* of an artificial neural network. The first simple perceptron looked like as shown in figure 3.5b.

McCullcoh & Pitts (1943) were trying to understand the neurons and they came up with the idea of the **perceptron**. The inputs could be 0 or 1, it could do operations like OR, AND, NAND, and NOT. However, people came to the conclusion that it was just a linear regression model and the machine couldn't be able to predict or learn with such a simple model.

In 1949, Donald Hebbs, a psychologist, claimed a theory that revolutionized the artificial neural network perception. With **Hebb's Theory**, nowadays, in DL, weights are applied in each perceptron for the inputs and are updated to make stronger and persistent models. This is called the learning process in DL.

In 1962, Rosenblatt, combined MacCulloch & Pitts perceptron's theory and Hebbs theory to create the first perceptron as we know today.

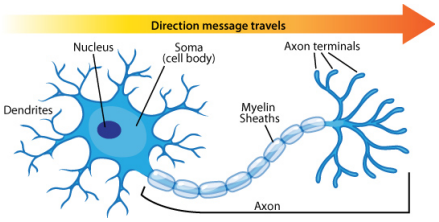
However, *Minsky and Papert (1969)* , mentioned in their book *Perceptrons: An Introduction to Computational Geometry* that Rosenblatt's model was nothing but a logistic regression and it would be

useful for creating a separation line between two classes but not on others like the exclusive-or (XOR) problems. This had a huge repercussion in people's interest in deep learning as they understood that these perceptrons weren't good at resolving AI problems.

It was not until the 1980's that better representation of the model was created which is today known as **forward propagation**.

The next section will explain:

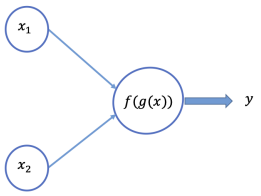
- What is a neural network?
- What is a **deep** neural network?
- What is forward propagation?



(a) A Neuron's Basic Anatomy.
Source: Arizona State University

Threshold Function

$$y = f(g(x)) = \begin{cases} 1, & g(x) \geq 0 \\ 0, & g(x) < 0 \end{cases}$$



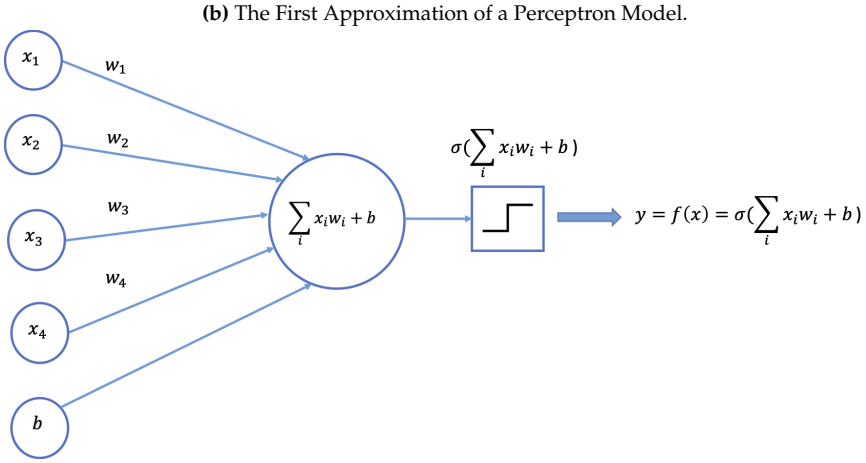
AND Function

Input x_1	Input x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR Function

Input x_1	Input x_2	Output
0	0	0
0	1	1
1	0	1
1	1	1

(b) The First Approximation of a Perceptron Model.



(c) Rosenblatt's Perceptron Model.

Figure 3.5: History of perceptron.

3.4 Neural Networks

In the previous section, perceptron was introduced, similar to the brain's neurons, perceptron needs to be connected to other perceptrons in order to create a network and exchange information between them. This is how neural networks create non-linear or more complex models.

In figure 3.6a, there are two simple linear perceptrons and both model's outputs are also illustrated.

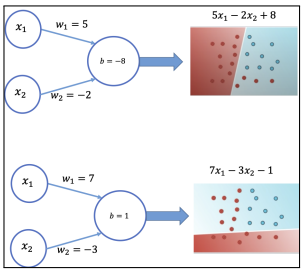
Next, in figure 3.6b, the two linear functions are added together to create a non-linear function and a neural network has been created.

Figure 3.6c is one form of representing the neural network and the 3.6d, is another where the activation function sigmoid is represented instead of the bias as in the other one.

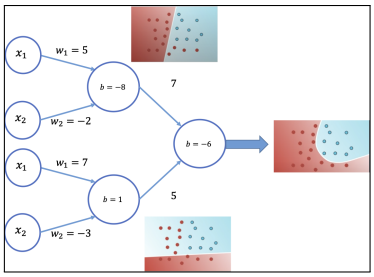
Activation functions will be seen in detail in the section 3.6, however, in this case, by combining two linear models, the probability of both models, gives something greater than 1, to adjust the value between 0 and 1, a sigmoid function is needed. In the example above in McCullcoh & Pitts' and Rosenblatt's model, a threshold function was used as an activation function as shown in figure 3.5.

Finally, a NN can be seen as illustrated in figure 3.6e with its input, hidden and output layer.

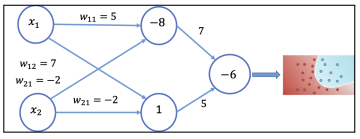
This whole chapter is dedicated to explaining DL as this technique was used to design and create the ASR.



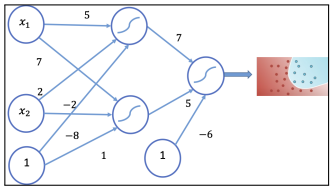
(a) Simple Linear Perceptron Examples.



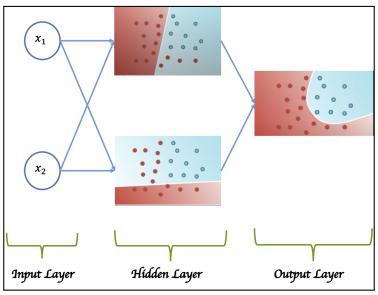
(b) Combining Linear Perceptrons.



(c) Neural Network Representation 1 with Bias.



(d) Neural Network Representation 2 with Activation Function.



(e) A Neural Network.

Figure 3.6: From Perceptron to Neural Network.
Source: Udacity Deep Learning Course Notes.

3.5 Deep Neural Networks

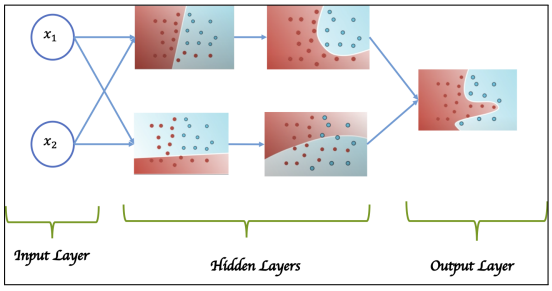
In the previous section, a neural network was illustrated and shown what can be achieved by combining different perceptrons. This section will cover what does it mean by a **deep** neural network.

Figure 3.6e illustrates a neural network with only a hidden layer. In Deep Neural Networks (DNN), the massive number of hidden layers is what makes it **deep**. This is also known as a multi-layer perceptron or MLP.

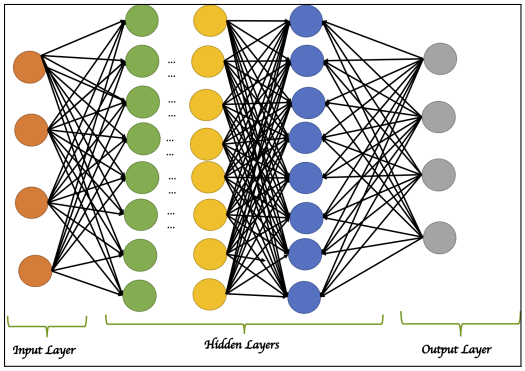
The model's complexity is proportional to the depth of hidden layers. An example can be seen in figure 3.7a. And a generalized architecture of a DNN would be the one illustrated in the figure 3.7b.

The most known DNN architecture is **ImageNet** [35]. The input of this model takes a RGB image of size 256x256. The hidden layer is composed of total eight layers, the first five are composed of convolutional layers and then followed by three dense layers. The output layer is a classification layer that maps the output to 1000 types of images. The neural network is composed of 60 million parameters and 650,000 neurons [35]. Graphic Processor Units were used to train the network efficiently.

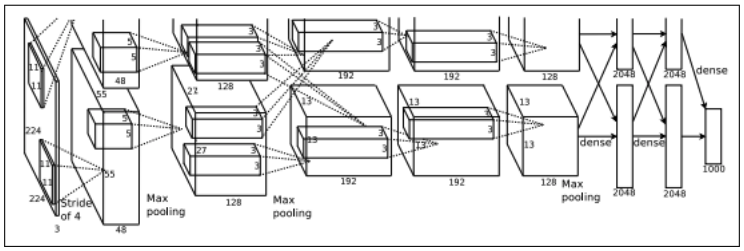
Imagenet is a multi-class classification problem as it classifies what the image is among 1000 options of different images. For this type of outputs, a **softmax** is used as the activation function in the last layer.



(a) Simple Deep Neural Network Example.
Source: Udacity Deep Learning Course Notes.



(b) Deep Neural Network Generalized Architecture.



(c) ImageNet Classification Architecture
Source: Original picture from the paper *ImageNet Classification with Deep Convolutional Neural Networks [35].*

Figure 3.7: Deep Neural Networks.

3.6 Activation Functions

As seen in some examples in the previous sections, the activation functions are used to create the output of the neural networks; for classification (softmax), for binary classification or for taking decisions (threshold).

Activation functions are the vital piece of neural network and without them, the output of the neural networks cannot be calculated precisely and efficiently. They are mathematical functions applied to create those desired outputs.

As seen in figure 3.5c, the mathematical equivalent of a perceptron (a simplified version of a neural network) is as shown in the equation 3.1.

As seen in section 3.4, the inputs are fed in the input layer and they are multiplied by the weights of each neuron, a bias (or an offset) is added to this sum. As the last step, all the results of the previous steps have to be passed through an activation function in order to select the neurons that should be activated to create the desired output.

$$y = \sum_{n=i} x_i w_i + b \tag{3.1}$$

According to the book [34] written by Ian Goodfellow, a very important figure in deep learning, deep neural networks are more precise as more hidden layers are introduced in it. If so, then the efficiency and time of the calculation will also increase. Activation functions play an important role in the efficiency of the neural networks as they reduce the calculations dramatically.

Activation functions are mainly for:

- Introducing non-linearity in the NN.
- Making the network more efficient as it normalizes the input between 0 and 1 and thus, making the values consistently smaller.
- For all the above, activation function trains the model efficiently.

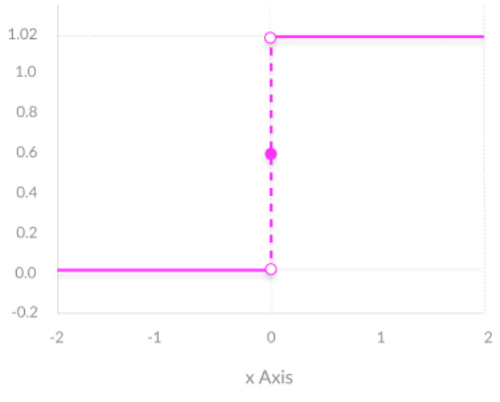
In this section, the most common activation functions will be described.

3.6.1 Threshold function

The **threshold function** is the most simple function of all and its mathematical representation is as shown in equation 3.2.

It normalizes all the values between 0 and 1. It is only a binary classifier and cannot solve for example an image classification problem.

Another problem with threshold function is what happens if we have a real small value like 10^-9, it will also be classified as 1, although its probability is tiny.



$$f(x) = \begin{cases} 1, g(x) \geq 0 \\ 0, g(x) < 0 \end{cases} \tag{3.2}$$

Figure 3.8: Step or Threshold Function.
Source: Missing Link AI

3.6.2 Sigmoid Function

The sigmoid function is a non-linear logistic function, which means that it normalizes the neuron's values between 0 to 1. Hence, the sigmoid function is also called the *squashing* function. It is used mainly for models to predict the probability of the output.

It is a differentiable function, which comes in handy for the gradient calculation in deep learning (discussed in sections below).

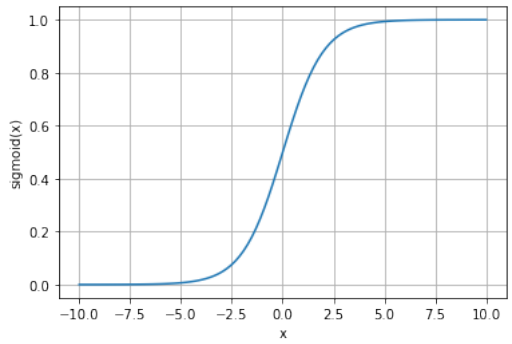


Figure 3.9: Sigmoid Function.
Source: My Jupyter notenook $neural\_networks.ipynb$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{3.3}$$

3.6.2.1 ReLU Function

ReLU stands for **Rectified Linear Unit** and without any doubt, this is the function that is used the most for hidden layers.

This function retains the positive values and discards the negative ones. For example, all the positive values of an image will be retained while the negative pixel values will be turned 0, which is the color black as a pixel.

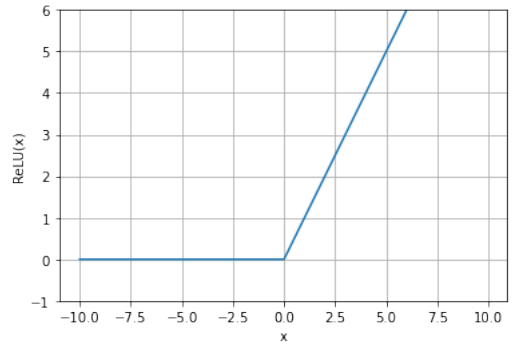


Figure 3.10: ReLU Function.
Source: My Jupyter notenook $neural\_networks.ipynb$

$$ReLU(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases} \tag{3.4}$$

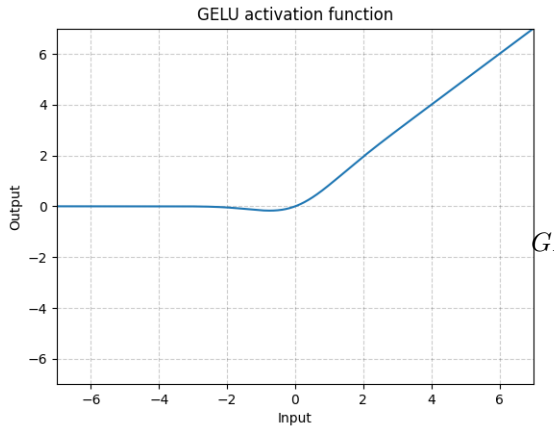
3.6.3 GELU - Gaussian Error Linear Unit

This is not a commonly used function and its use has started recently. GELU function was introduced on 2016 by Dan Hendrycks and Kevin Gimpel in the paper *GAUSSIAN ERROR LINEAR UNITS (GELUS)* [36]. Their statement in the paper is very clear:

"We perform an empirical evaluation of the GELU nonlinearity against the ReLU and ELU activations and find performance improvements across all considered computer vision, natural language processing, and speech

tasks." [36]

GELU is used in Google's Bidirectional Encoder Representations from Transformers, mostly known as, BERT [37]. It is a seach algorithm which understands more natural language.



$$GELU(x) = 0.5x(1 + \tanh\,(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3))) \tag{3.5}$$

Figure 3.11: GELU Function.
Source: GELU- Pytorch Master Documentation

3.6.4 Softmax Function

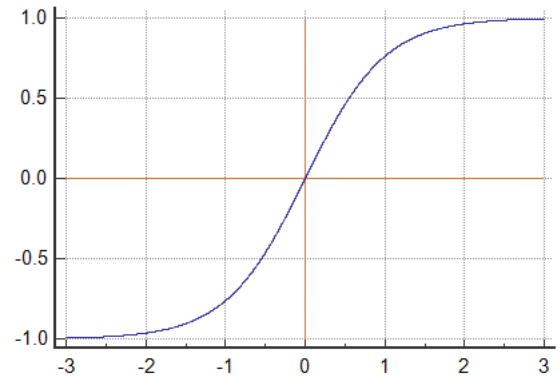
Softmax function is used in classification problems. It is commonly used only in the output layer of the neural network.

In this project, softmax function is used at the end of the neural network to calculate the probability of the output being one of the 28 possible values, that is 26 possible alphabets (ñ is not included), space and empty value.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))} \tag{3.6}$$

3.6.5 The Hyperbolic Tangent Function

The tanh function looks very similar to the sigmoid function, it is often compared to the sigmoid function. It limits the inputs values between [-1, 1], it makes the values smaller.



$$\tanh\,(h) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.7}$$

Figure 3.12: tanh Function.
Source: MEDCALC

3.7 Feedforward Neural Network

Also known as, FFNN, this is the process how the network learns to create the desired output and this section will cover the topic with mathematical calculations for a better understanding.

To keep it simple, it will be assumed that the neural network has only one hidden layer and if the neural network is seen as a black box then it can be illustrated as shown in figure 3.13a. In the mentioned figure, there are n inputs, k outputs, and w is weight of the neurons.

The neural network is static, which means that for the same inputs and same weights, it produces the same outputs. Only if the weights are changed or the inputs are changed then the outputs will be changed. In the neural networks, normally the weights are initialized randomly or sometimes even initialized as zeros. The networks task is to learn to update those weights to create the desired outputs.

The NN has to pass two phases:

1. **Training:** In this phase, the neural network is fed with inputs and the labelled outputs; the model is trained to create the desired output. For example, for image classification, the neural network, the inputs would be the images and the labels would be what the image is.
2. **Evaluation:** New inputs will be introduced to the **trained** network and it should be able to create the desired output. For example, in the image classification, if a new image not used in the training phase is introduced then the trained network should be able to classify it.

It can happen that the network in the training phase, **overfits** the training data. Which means that it performs perfectly on the trained data as expected but doesn't perform as expected on a new data or input. There are techniques to avoid this and will be seen later.

The figure 3.13b, is a neural network with n inputs, m hidden neurons and k outputs. As shown in the equation 3.8, each input is connected to the hidden neurons multiplied by a set of weights ${W_{ij}}^1$. Additionally, each hidden neuron is connected to each output multiplied by another set of weights ${W_{ij}}^2$.

The goal of the neural network is to yield the correct outputs by updating the weights and achieve the optimal ones that provides the desired results and this is done in the training phase.

Training Phase

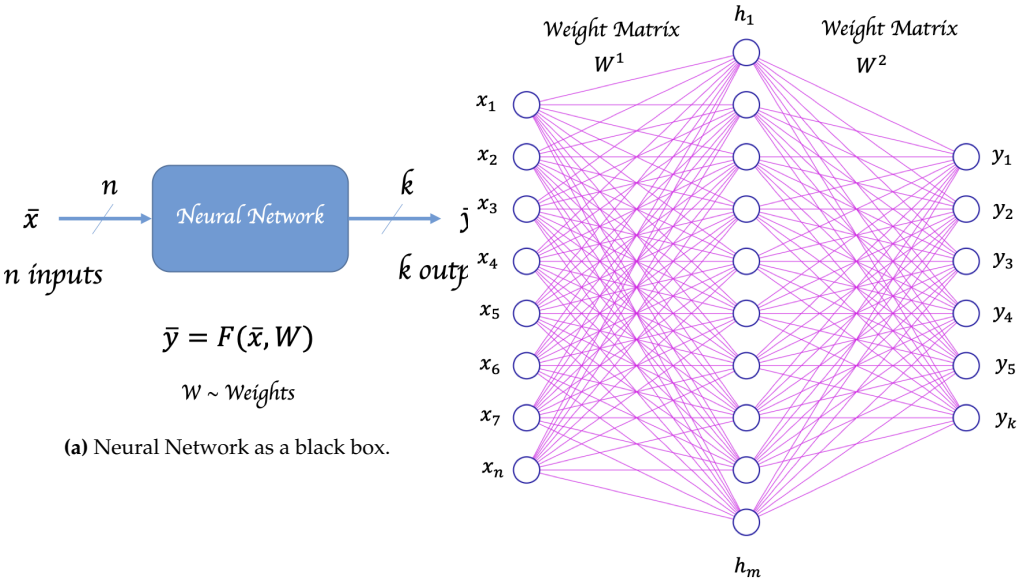
The training phase consists of the neural network iterating as many times needed to get the optimal weights. In the iteration, the neural network has to follow two steps:

1. **Feedforward:** The output will be calculated and then it will be compared to the correct label. This comparison will provide the error in the calculation/prediction of the output.
2. **Backpropagation:** After obtaining the error, the neural network has to go backwards and minimize the error and thus calculate new weights to produce new outputs.
3. **Iterate:** The neural network performs step 1 again which is the feedforward and then again backpropagation. Normally, the number of iteration depends on data volume, number of hidden layers, complexity of the neural network and has to be determined empirically.

The methods of finding h and y will be discussed in the following section.

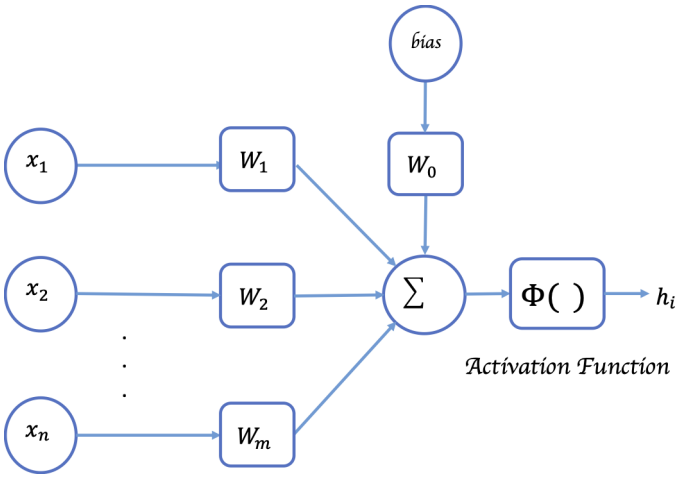
$$h_i = F(x_i, {W_{ij}}^1) \tag{3.8}$$

$$y_i = F(h_i, {W_{ij}}^2) \tag{3.9}$$



(a) Neural Network as a black box.

(b) Neural Network with a hidden layer.



(c) A closer look of the neural network.

Figure 3.13: Neural Network Representation in different ways.

3.7.1 Feedforward

Feedforward is the step 1 for finding the optimal weights and the desired outputs.

Take as reference the figure 3.13b, limit the number of hidden neurons to 3 so, $m = 3$ and also limit the number of outputs so, $k = 2$.

1. **Finding the hidden neurons values:** Until these values are not the optimal values, it will be denoted as h'.

   In the equation 3.10, the W_{ij}, corresponds to the weight W from the input x_i to the neuron h_j. So, this is what is reflected in the figure 3.13c, until the sum box. Now, the next step for the feedforward is to apply the activation function.

$$h' = (xW^1) \tag{3.10}$$

$$\begin{bmatrix} h'_1 & h'_2 & h'_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & ..... & x_n \end{bmatrix} . \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ . & . & . \\ . & . & . \\ . & . & . \\ W_{n1} & W_{n2} & W_{n3} \end{bmatrix} \tag{3.11}$$

   An activation function is used to identify the non-linearity between the inputs and the outputs; and also to avoid huge numbers in the calculation as already seen in the previous section, different types of activation function limit the outputs in a different way. For example, in ReLU, all the negative values are nulled and the positive values will retain their original value.

   Now with the activation function ϕ, the equation 3.10 can be re-written as shown in the equation 3.12.

$$h = \phi(xW^1) or h = \phi(h') \tag{3.12}$$

$$h_1 = \phi(x_1W_{11} + x_2W_{21} + ... + x_nW_{n1}) \tag{3.13}$$

$$h_2 = \phi(x_1W_{12} + x_2W_{22} + ... + x_nW_{n2}) \tag{3.14}$$

$$h_1 = \phi(x_1W_{13} + x_2W_{23} + ... + x_nW_{n3}) \tag{3.15}$$

2. **Finding the output values:** As there will be more than one output, y will also be a vector. Mathematically, the idea is the same as seen in the previous step.

   So, for the outputs' values, the mathematical expression will be as shown in the equation 3.16. Please note that in this case, the weight matrix has 3 rows (3 hidden layers) and only 2 columns (2 outputs).

   In the case of outputs, an activation is not necessary. In the case of classification problems, the softmax activation function is used as shown in equation 3.6.

$$y = h.W^2 \tag{3.16}$$

$$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} . \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix} \tag{3.17}$$

In reality, to get a good approximation of the outputs, the hidden layers needs to be deep and the equations seen above will be the same but for larger dimensions.

Take as reference the figure 3.13b, then the general mathematical expression will be as shown in the equation 3.18. If there are more than one hidden layers than the previous hidden layer will be the input.

$$h_1 = \phi(\sum_{i}^{n} x_i W_{i1}) \tag{3.18}$$

$$h_2 = \phi(\sum_{i}^{n} x_i W_{i2}) \tag{3.19}$$

..... (3.20)

$$h_m = \phi(\sum_{i}^{n} x_i W_{im}) \tag{3.21}$$

(3.22)

So, going back to the goal which is to find the optimal weights to get the desired outputs, when the output is calculated by the neural network, it compares this value with the desired output.

For example, if there is a dataset of images with its corresponding labels (a cat's image is named as a cat and a dog's image is named as a dog, etc.), at the end of each feedforward the neural network calculated the **error value** by comparing the output calculated and the labeled value. If the neural network's output was a fish but the input image was a dog then the error is high. The neural network will then do the backpropagation to minimize this error.

3.7.2 Backpropagation

As mentioned in the previous section, the neural network will have to minimize the error value, which is basically the difference between the desired output and the calculated output. This is also called the **loss function** and is expressed as the equation 3.23, where d is the desired output and y is the calculated output.

$$E = (d - y)^2 \tag{3.23}$$

Backpropagation is the process of going from the outputs towards the inputs to update the weights to decrease the error. This is essentially, stochastic gradient descent calculated using the chain rule.

The figure illustrates a neural network's weight and error. If in one of the iterations is point A, to reduce the error E_A, the weight W_A needs to increase as the minimum is on its right-hand side. The orange arrow represents the gradient or the derivative of the slope in the point A. In this point A, the gradient is negative.

Now, if the same analysis is done for the point B, the gradient at this point is positive. This method of optimizing the weights is called the **Gradient Descent**.

This allows the conclusion that:

- to increase the weights, the gradient is negative.
- to decrease the weights, the gradient is positive.

For a determined weight W, the new weight calculated using the backpropagation theory would look as shown in equation 3.24, where α is called the **learning rate**. The learning rate determines the speed towards the optimal weights, it basically defines if the training steps are moving fast or slow to achieve the optimal weights which is the main goal.

Learning rate is one of the parameters, in deep learning called hyperparameters, that needs to be tuned once the model has been designed. If the learning rate is too large, the optimal weight can be

skipped, if the learning rate is too small, then smaller steps have to be taken and thus, slowing down the learning process.

In the equation 3.24, partial derivative is used because the error depends on different weights and not just one weight. Looking again at the figure 3.13b, the outputs depend on the weight matrices W^1 and W^2. The partial derivative $\frac{\partial E}{\partial W}$ is needed to know the impact of all the parameters in the error.

Please note that the optimization of the weights using the backpropagation algorithm is called **Stochastic Gradient Descent** or oftenly known as SGD.

$$W_{new} = W_{previous} + \alpha(-\frac{\partial E}{\partial W}) \tag{3.24}$$

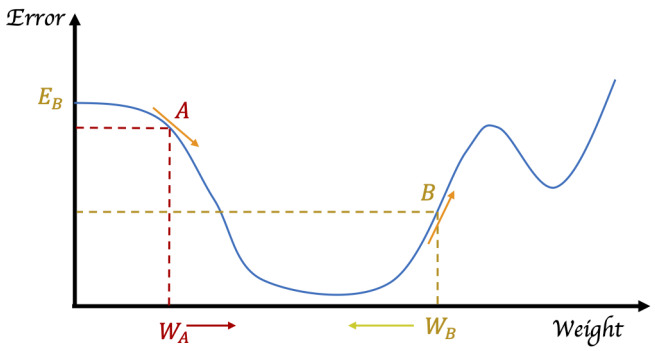


Figure 3.14: Error, Weight & Gradient.

3.7.3 Error or Loss Function

The error function calculates the difference between the goal and the calculated output. The most used error functions in deep learning are:

1. **Mean Squared Error:** Mostly known as MSE, a metric used for evaluating the prediction. Vastly used in regression models and is computed as shown in equation 3.25 where d is the desired output and y is the predicted one.

$$MSE = \frac{1}{n}\sum_{n=1}^{k}(d-y) \tag{3.25}$$

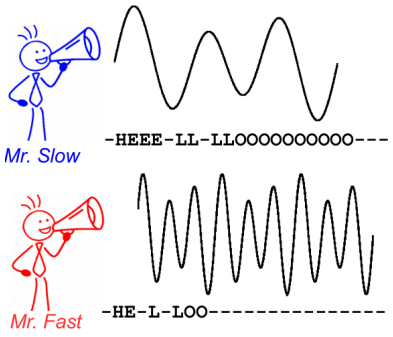
2. **Cross Entropy:** This is used in classification problems and is formulated as shown in the equation 3.26. A higher cross-entropy implies a lower probability of an event.

$$Cross - Entropy = -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \ln p_{ij} \tag{3.26}$$

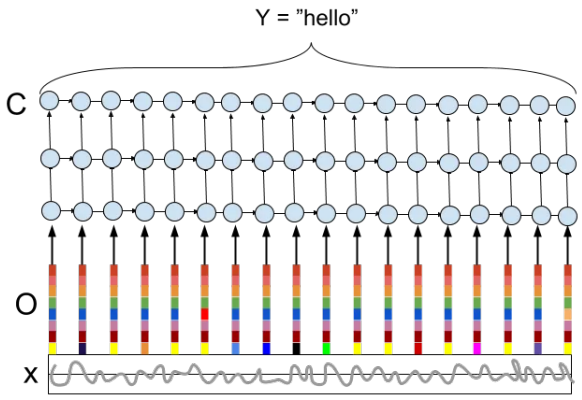
For this project, the loss function, **Connectionist Temporal Classification** [19], also known as CTC.

Imagine, labeling per [ms] word by word or phoneme by phoneme the audio to make the prediction. It is actually impossible to do so for thousands and thousands of audio data. It can become tedious, boring, and time-consuming to annotate that volume of data on character-level.

Even if it is labeled that way, the same character can occupy more than one time-step, see figure 3.15a. There can be people who talk slower or those who talk faster.



(a) Mr.Fast and Mr.Slow Example.
Source: Stack Exchange.



(b) CTC with the Sotmax probabilities for each character at a given time t.
Source: Knowledge Transfer.

Figure 3.15: Connectionist Temporal Classification examples

Here is how CTC works:

- First, it calculates the probability of each time step of being a certain character from a bank of vocabulary as shown in figure 3.15b.
- However, it doesn't know the alignment of the final output so, if Mr. Slow was to be speaking then it would be something like -HEEE-LL-LLOOOOOOO— but the final output is actually *HELLO*, so, at first glance, the easy solution would be to delete the repetition but then the output would be *HELO* and not *HELLO*.
- So, CTC introduces what is called a **blank** and is represented by **-** (which is not visible in the final output. Whenever there is this blank, the repetitions are kept. For example, in the figure 3.15a:
  - Mr. Slow, would be -HE-L-L-O- and finally HELLO.
  - Mr. Fast, would be translated as -HE-L-LO and finally HELLO.

3.7.4 Optimizers

The loss function mentioned in the section above is the function that measures if the weight optimization is going in the right direction or not.

Optimizers are the ones responsible to make the model the more accurate possible by optimizing the weights of the neural network. As seen before, gradient descent is one of the most famous methods used. The equation 3.24 was specific for gradient descent, if generalized for any other optimizer, it would look like the equation 3.27 where:

- θ represents the weights, bias, and activation function.
- α is the learning rate, which is explained in detail in the next section.
- ∇ is the gradient of the function J.
- J is the function to optimize.

$$\theta = \theta - \alpha.\nabla_\theta J(\theta; x, y) \tag{3.27}$$

Imagine a man trying to descend a mountain but can't look where he's going, he will just have to keep on descending without looking until he reached what he thinks is the lowest point of the mountain and can't descend anymore. However, it can happen that the man has reached a valley and actually isn't the lowest point of the mountain.

This would be a local minima and while looking for the optimal weights, the neural network can get stuck in a local minima thinking that it has reached the optimal weight(global weight). To avoid this, **momentum** can be used. Taking the same man as example, each time this man falls in a minima (unknown if local or global), he will calculate the average of his previous steps but assigning weights to these steps, to the recent one, he can assign 1, to the previous one β and to the previous $\beta - 1$ and so on. **Momentum** accelerates the speed of the learning procedure. It can be represented as in the equation 3.28, where v_t is the last update to θ.

$$\theta = \theta - \alpha.\nabla J(\theta_t) + \alpha.\sum_{n=1}^{t} \nabla J(\theta_\tau) \tag{3.28}$$

Adaptive Gradients [38], known as Adagrad is also another popular optimizer and has demonstrated to perform well on distributed or sparse data. Learning rate is one of the most hyperparameter and choosing a bigger or a smaller learning rate impacts the training of the neural network.

Adagrad uses dynamically the knowledge gained from the geometry of the parameters from the previous iterations to scale the learning rate. So, if any feature is less common (in speech, for example, *predecessor* is a word with less occurrence than the word *the*), Adagrad highlights these features by setting higher learning rates to these and lower learning rates to more frequent features.

Root Mean Square Propagation, was first introduced by **Geoffrey Hinton et al** [39]. The problem addressed is that when updating the weights by using the mini-batch algorithm, what if a mini-batch's gradient is larger than the successive one? If the mini-batches don't have similar gradients then there would be a drastic variety in increments and decrements in weights. RMSProp uses a running average of recent gradients to update the weights.

Adaptive Moment Estimation, mostly known asAdam [40], is the best optimizer until date and it was discovered in 2015. It combines momentum ¡, Adagrad and RMSProp. This is the one used in this project and in pytorch it is very easy to use.

3.7.5 Hyperparameters

The hyperparameters are the parameters that are external to the model and needs to be tuned by the practitioner. There are some rules that can be followed but from personal experience, it is more of a trial and error result.

The hyperparameters are variables that need to be determined before starting the model training.

3.7.5.1 Learning Rate

The learning rate is the most important parameter. Even if someone has a perfectly fit trained model and another person uses it with a different dataset, it is probable that more than one learning rate has to be tried before the model is trained properly.

Learning Rate is what was explained in the 3.7.2 section. It is the step that the model will take in each iteration to get near the minimum error and thus get the optimal weight.

How to know if the learning rate used in the model is good or not?

- **Good Learning Rate:** The validation error is decreasing during the training.

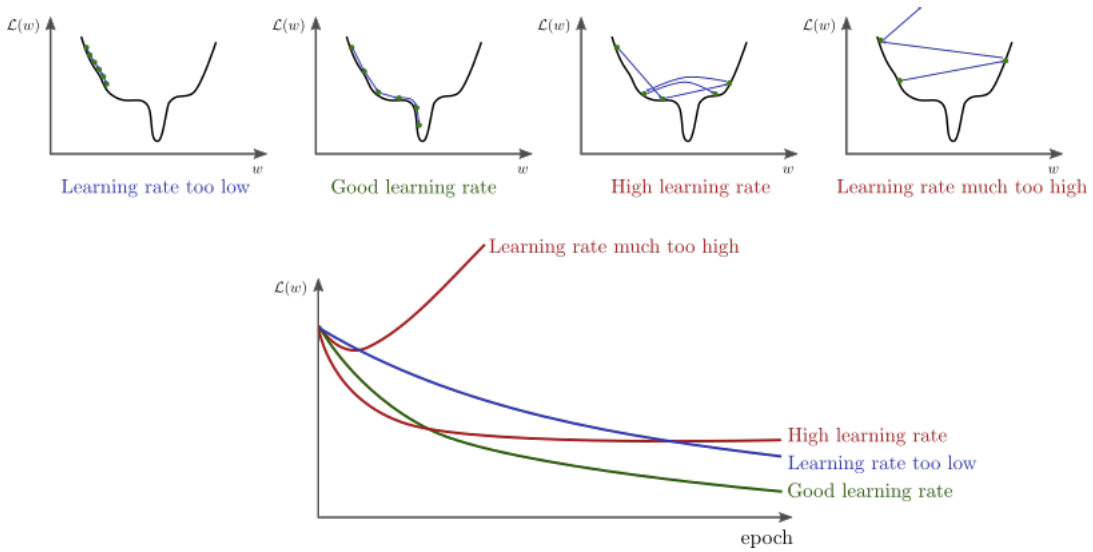


Figure 3.16: Learning Rate Cases.
Source: BD Hammel - AI Researcher

- **Very Small Learning Rate:** If the learning rate is too small, the steps are so small that the search of the minimum error will not be efficient. The validation error is decreasing too slowly during training. A larger learning rate is needed.

- **Too Large Learning Rate:** If the learning rate is too huge, it may never get to the minimum error, it may even surpass the minimum error and never get there. The validation error is increasing during training. It needs to be decreased.

The figure below 3.16 illustrates the different cases of the learning rates election.

There are many ways to assign a learning rate, as seen in the previous section, the learning rate is actually very related to the optimization function. Furthermore, besides the option of being a fixed value, it can also be used as a **learning scheduler**. There are many types of learning schedulers but for this project, only one was used, which is called the **One Cycle Learning Rate** [41]. The idea is to create a cycle of two steps and in one step, increment the learning rate from the lowest to the highest learning rate and in step 2, decrease the highest learning rate to the lowest again. By doing this, overfitting is avoided.

The next section explains what are these iterations, how it is called, and how to tune it.

"This is often the single most important hyper-parameter and one should always make sure that it has been tuned" - Yoshua Bengio [42]

3.7.5.2 Training Iterations/Epochs

The training iteration, known as a **epochs** is the iteration of feedforwarding, calculating the loss or the error, backpropagating, and optimizing the weights for all the dataset.

The validation error is what determines the number of epochs. When the validation error has stopped decreasing after a particular number of epochs then the training should be stopped.

There is a technique called **early stopping** which will be explained in the next section and is also used for solving other problems.

3.7.5.3 Batch Size

The **batch size** is a hyperparameter that is used during the gradient descent phase. There are three types of batches.

To create a batch, the training dataset is divided into one or more batches. During the optimization of the model, the dataset is processed in batches.

There are three types of batches:

1. **Batch Gradient Descent:** This is when the batch size is the same as the total length of the dataset. The computational cost of this is very high. Also while calculating the gradient descent on the whole dataset makes the gradient descent much higher.
2. **Stochastic Gradient Descent:** When the batch size is 1, this means that only one sample of the dataset is taken for the optimization at once. Or in other words, for every sample, the gradient descend is calculated and the movement towards the optimal weight is very slow. The movement is also noisy because the noise is more noticeable if each sample is taken at once. The noise is smoothed if the average of it is taken as the following one.
3. **Mini-Batch Gradient Descent:** This is when the batch size is greater than one but less than the length of the dataset. Normally, the values to choose are binary numbers like 32, 64, 128, 256, etc. The gradient descent is updated only after a few samples, like 32, 64, 128, etc. There still is noise but it is reduced.

The chosen batch size affects the training efficiency as in computational cost. In the project and normally, a good starting choice of batch size is 32, when the model is too complex or the computation power is not good, the batch size has to be reduced.

3.7.5.4 Number of Hidden Units and Hidden Layers

According to *Andrej Karpathy* [43], in practice, a 3-layer neural network outperforms a 2-layer neural network. This is not always true, for convolutional neural network, the more deeper the NN, the better.

This is against what *Ian Goodfellow* states in his Deep Learning book [34] as he says that the deeper the network, the more precise.

Actually, this hyperparameter as the other is also tuned by trying the model and comparing the results.

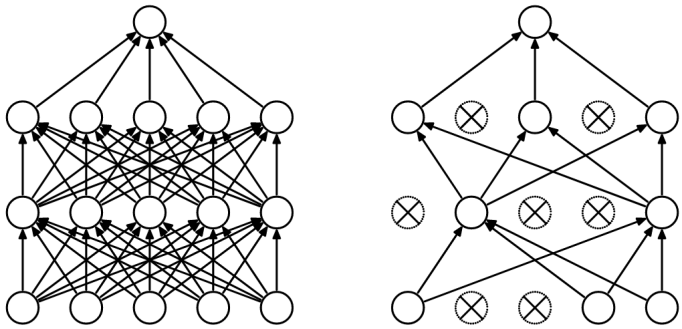
The choice of number of hidden units or neurons impacts what is called the overfitting and the underfitting problem described in the section 3.7.6.

If the model starts overfitting then the number of the hidden neurons needs to be reduced. If the model is underfitting then the number of the hidden neurons needs to be increased.

3.7.6 Overfitting and Underfitting

Once the model is trained, it needs to be **evaluated** with a different dataset other than the training dataset to know how the model is performing. This is done by taking a subset of the training dataset and creating a validation dataset. In each iteration, when the training is done in the model for the training dataset, an evaluation is done with the validation dataset to know the errors or how the model is performing.

In general, the available dataset is broken into three dataset:



(a) Standard Neural Net (b) After applying dropout.

Figure 3.17: Dropout image from the original paper *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* [44]

1. **Training Dataset:** It is used during the training process to update the model weights.
2. **Validation Dataset:** This is used during the training process to check how well the model generalizes.
3. **Test Dataset:** It is used after the training with the trained model to check the accuracy of the model.

This is done to avoid problems that can occur: Overfitting and Underfitting (below).

3.7.6.1 Overfitting

Overfitting occurs when the model doesn't **generalize** well enough. This means that during the training process, the model has fit too well to the training data. When new input is introduced to the trained model, this model will not perform as good as it performed with the training data.

For example, it is like studying to be able to answer the questions but once in real-world or for any other purpose not being able to answer questions related to the subject.

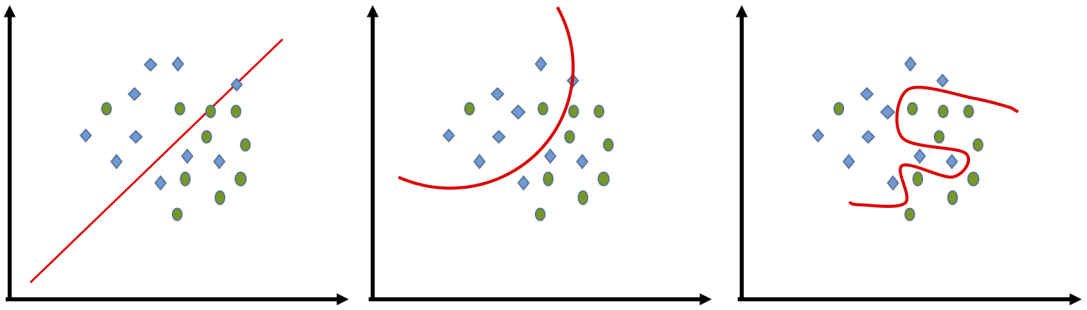
It can happen that the model is too complex and the volume of the training dataset is too small. It can also occur because the model is too complex for the problem in hand and a simpler model would work better.

Overfitting is detected when the training error is much lesser than the validation error. The overfitting problem can be solved by using two techniques:

- **Early Stopping:** As already mentioned before, overfitting is detected when the training error is lesser than the validation error. To use the early stopping, the running epoch's validation loss is compared to the previous one and if it is smaller then overfitting should be considered. Both training loss and validation loss doesn't decrease smoothly, so the training shouldn't stop at the first detection of validation loss increase. It should be stopped after a particular number of epochs.
- **Dropout:** Dropout [44] is defined by a number which is a probability, normally, 0.2. This means that to avoid overfitting, for every feedforward and backpropagation, each hidden layer neuron won't be considered (will be dopped out) with a probability of 20% in each epoch.

3.7.6.2 Underfitting

Underfitting is just like going to an exam without having studied anything.



(a) Underfitting Model. **(b)** Robust Model. **(c)** Overfitting Model

Figure 3.18: Scenarios after training a model.

In DL, the model underfits because the model is unable to reduce the training loss because of the model simplicity. This can happen because a simpler model is unable to spot the features that needs to be extracted from the data.

Usually, when underfitting occurs, the model is unable to reduce the training loss because it isn't able to optimize the weights.

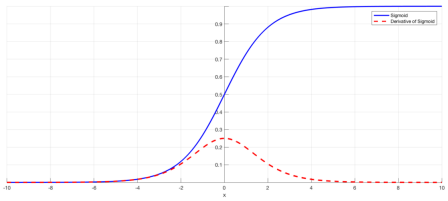
As mentioned earlier, to avoid underfitting, the number of hidden neurons should be increased or the model should increase in complexity.

3.7.7 Vanishing Gradient

As seen in the section 3.7.2, to optimize the weights, the partial derivatives has to be calculated for the whole neural network, starting from the output layer to the input layers.

This means that even the derivatives of the activation function has to be calculated. Take for example, the sigmoid function 3.9, the derivative of the sigmoid function is as shown in the figure 3.19. The derivative tends to be smaller, at each backpropagation, this value will become tiny, so tiny that the weights will be approximate to zero. This problem is called the **vanishing gradient**.

The solution to the vanishing gradient is simple, changing the activation function. Take for instance the ReLU function, 3.10, the derivative of the ReLU function is 1, this is why, it is used vastly in deep learning.



$$\sigma\prime(x) = \frac{1}{1+e-x}.(1-\frac{1}{1+e^{-x}}) \qquad (3.29)$$

Figure 3.19: Derivation of Sigmoid Function
Source: Derivative of the Sigmoid function

3.8 Summary

In the previous sections, all the steps and concepts were introduced for a better understanding of deep neural networks. Although, the explanation was done for a simple model, the steps and the concepts apply for any type of neural networks.

These are the steps to follow for training a DNN:

1. Load the data.
2. Define a model.
3. Specify the loss function and the optimizer.
4. Train the network:
   (a) Define the number of epochs.
   (b) Keep track of the training loss.
   (c) Create a loop to iterate over every mini-batch:
      i. Clear all accumulated gradient.
      ii. Forward pass. Introduce the input to the model, calculate the output.
      iii. Calculate the loss.
      iv. Backpropagate and optimize the weights.
      v. Save the training loss.
      i. Turn on evaluation mode.
      ii. Iterate through the validation data.
      iii. Forward pass the validation data.
      iv. Do not do backpropagation.
      v. Keep track of the validation loss.
5. Once the model is trained, it needs to be tested with the test dataset and check the test acuracy.

3.9 Recurrent Neural Networks

Recurrent Neural Networks, also known as RNN, are used for inputs with time dependencies or are sequential data. In the real world, many applications have temporal dependencies, for example, Alexa, Siri, automatic subtitling, predicting the next word in a sentence, etc.

RNN are based on the same basics of the FFNN. In RNN, not only the current input is taken into account as seen until now, but it also takes into account the inputs from previous time steps.

In FFNN, each input was considered independent, as an image of a cat, an image of a dog, or a frog. The relationship between these was not considered at all. RNN were created thinking of representing time in the neural networks by *Jeffrey L. Elman* [45].

The figure 3.20 illustrates how the **Recurrent Neural Networks** are represented.

In the folded model, the network has a new element (compared to FFNN), which is S. It stands for state, a term used for systems with memory. This layer feeds itself back again, that is because, for each timestep, the system will look the same. All the other elements' definitions can be found below:

- W_x represents the weight matrix connecting the inputs to the state layer.
- W_y represents the weight matrix connecting the state layer to the output layer.
- W_s represents the weight matrix connecting from the state from the previous timestep to the one in the next timestep.
- x_t is the input vector at time t.
- y_t is the output vector at time t.
- s_t is the state vector at time t.

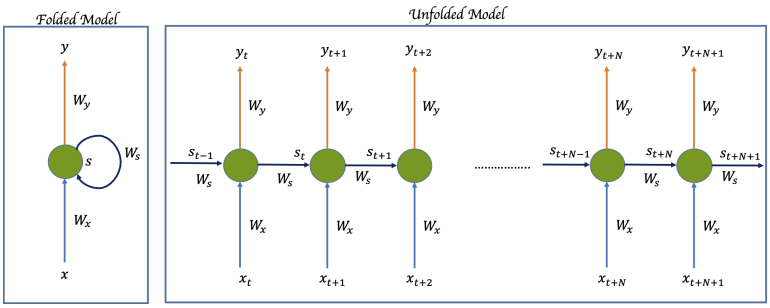


Figure 3.20: RNN unfolded and folded diagram.
Source: A deep learning framework for financial time series using stacked autoencoders and long-short term memory [46]

Look at the unfolded model in the figure 3.20, how will the activation function work here? Previously, for FFNN the equation for the calculation of the hidden layer was as shown in 3.18. In the unfolded model, given x_t, the calculation of s_t and y_t would be as shown in equation 3.30.

Similar to the FFNN, after the feedforward pass, for RNN needs to also optimize the weights. For RNN, this is called **Backpropagation Through Time (BPTT)**. The next section will explain how this is done.

$$s_t = \phi(x_t.W_x + s_{t-1}.W_s) \tag{3.30}$$
$$y_t = s_t.W_y \tag{3.31}$$

3.9.1 Backpropagation Through Time

This section will look into calculating errors and optimizing the weights to get the desired outputs. The error function now will be $E_t = (d_t - y_t)^2$ (a loss function) which is now dependent on time and needs to backpropagate through all the timesteps.

In the equation 3.32, it is obvious that to calculate the optimal weights, each and every parameters of the model has to be taken in consideration. Although, this seems like a heavy computational cost, take in account that for each timestep, all the parameters are the same. They will only change for a different epoch.

For reference and to understand better the equation 3.32 and 3.38 look at the figure 3.21. In the figure, the backpropagation can be clearly seen.

$$\frac{\partial E_3}{\partial W_y} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial W_y} \tag{3.32}$$
$$\frac{\partial E_3}{\partial W_s} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial W_s} \tag{3.33}$$
$$\frac{\partial E_3}{\partial W_s} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W_s} \tag{3.34}$$
$$\frac{\partial E_3}{\partial W_s} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W_s} \tag{3.35}$$
$$\tag{3.36}$$

Finally, the equation to adjust the weight W_s using BPTT can be described as shown in equation

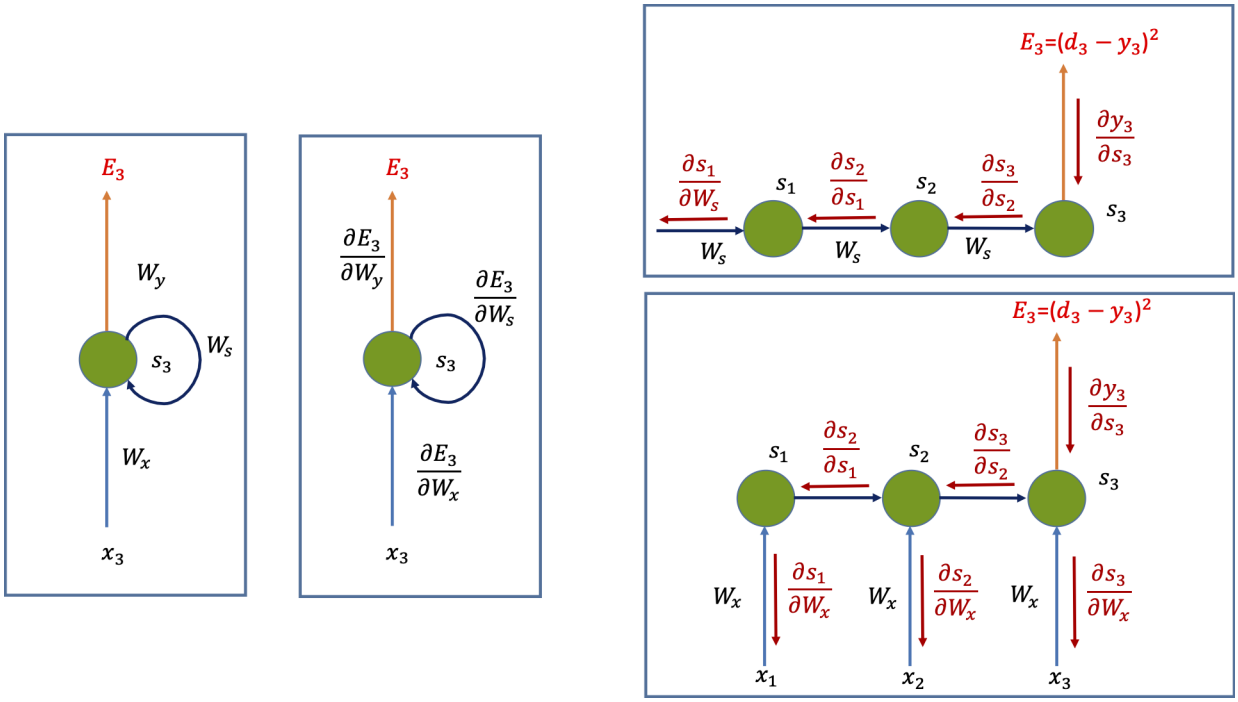


Figure 3.21: RNN Gradient Calculations to adjust W_s and W_x.

3.37 for any general RNN.

$$\frac{\partial E_N}{\partial W_s} = \sum_{i=1}^{N} \frac{\partial E_N}{\partial y_N} \frac{\partial y_N}{\partial s_i} \frac{\partial s_i}{\partial W_s} \tag{3.37}$$

To calculate the weights W_x, same logic can be applied and the BPTT would be as shown in the equation 3.38

$$\frac{\partial E_N}{\partial W_x} = \sum_{i=1}^{N} \frac{\partial E_N}{\partial y_N} \frac{\partial y_N}{\partial s_i} \frac{\partial s_i}{\partial W_x} \tag{3.38}$$

The main problem with this type of neural network is that if relationships need to be captured for more than 8 to 10 steps back, the vanishing gradient problem is inevitable and it is impossible to go back more than 8 to 10 steps. This model will only effectively perform for only inputs with very small timesteps.

3.9.1.1 Exploding Gradient Problem

The opposite of the vanishing gradient is the **exploding gradient** [47] where the value of the gradient becomes uncontrollably larger. To avoid this situation, the technique **Gradient Clipping** is used. A threshold is determined for the gradient, if it exceeds this threshold then it is normalized and thus, avoiding the exploding gradient problem.

3.9.2 Depth in Recurrent Neural Networks

Similar to the neural networks with more than one hidden layers, the RNN can also have more than one hidden layer. This type of networks are often known as **Deep Recurrent Neural Networks**.

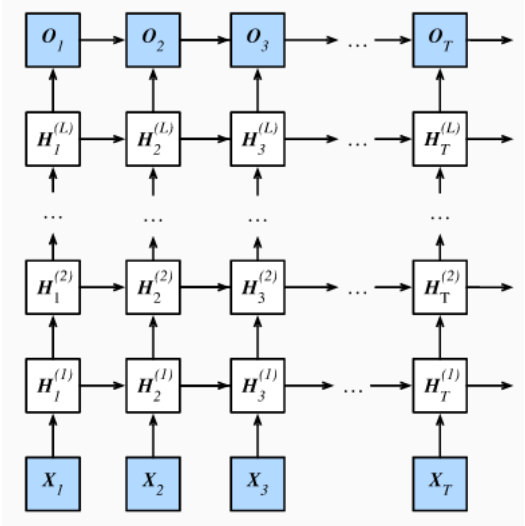


Figure 3.22: Deep Recurrent Neural Network.
Source: Dive Into Deep Learning.

3.9.3 Bidirectional Recurrent Neural Network

Bi-directional RNN are used for cases when both the present time t depends on the past and future elements in the sequence.

It can happen in speech or written text, a sentence can start somehow but may be the future additional information can provide context.

Bidirectional RNN are used for many applications. Below are listed some of them:

- Speech Recognition - *Framewise phoneme classification with bidirectional LSTM and other neural network architectures* [48].
- Speech Recognition - *Hybrid speech recognition with Deep Bidirectional LSTM* [49].
- Sentiment Analysis - *Trains a Bidirectional LSTM on the IMDB sentiment classification task [50]*
- Review Generation - *Review generation using Bidirectional Long Short Term Memory(LSTM)* [51]
- Medical - *Bidirectional RNN for Medical Event Detection in Electronic Health Records* [52].
- And many more.

The bidirectional RNN architecture is as shown in the figure 3.23. There is a split into two directions but the neurons from the positive direction doesn't have any interaction with the ones going in the opposite direction. However, the outcome of both of these neurons are considered for the calculation of the desired output.

3.9.4 Long Short-Term Memory

On 1997, Sepp Hochreiter and Jürgen Schmidhuber proposed the Long Short-Term Memory (LSTM)(LSTM) model to avoid the vanishing gradient problem in the RNN [53]. Since the introduction of LSTM, many applications have been developed. Here are some examples:

- Robot Control - *A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks* [54].

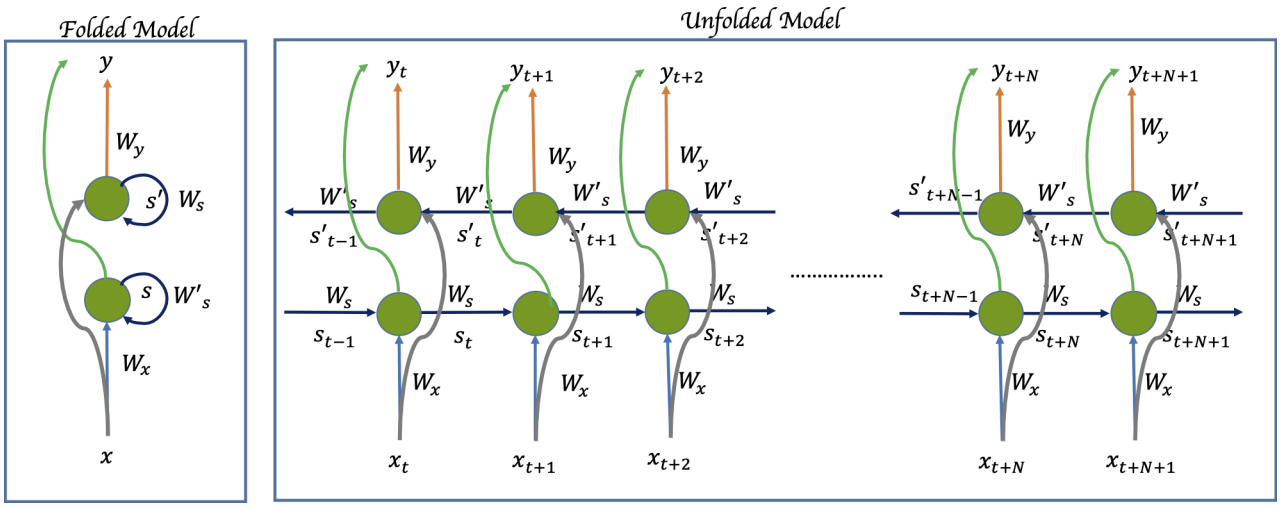


Figure 3.23: Bidirectional RNN architecture

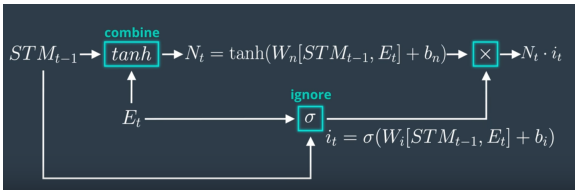
- Speech Recognition
  - *Speech Recognition with Deep Recurrent Neural Networks* with 17.7% test set error [55].
  - *Framewise phoneme classification with bidirectional LSTM and other neural network architectures* with accuracy of 78.1% in the training set. [48].
  - *An application of recurrent neural networks to discriminative keyword spotting* with keywork spotting accuracy of 84.5% [56].
- Music Composition - *Learning the long-term structure of the blues* [57].
- Handwriting Recognition - *Unconstrained Online Handwriting Recognition with Recurrent Neural Networks* [58]
- And many more.

The LSTM architecture is as shown in the figure 3.24d and it uses gates to predict the output. The gates are as follows:

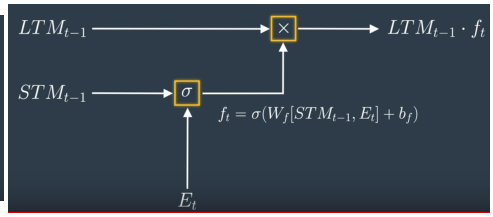
- **Learn Gate:** It takes in the short-term memory from the previous time STM_{t-1} and the event E_t as inputs and combines them both. The combination is done by a linear function with a weight matrix and adding a bias (just like in FFNN). Then the results are squashed by using the $\tanh$ activation function obtaining N_t. Not all of the N_t is stored, only the relevant information to the actual event is kept and the rest is ignored. It is done by multiplying a factor called the **ignore factor** i_t. This ignore factor is calculated by using a small neural network with a sigmoid function as the activation and using the information from STM_{t-1} and the event itself E_t as the inputs and thus get the i_t. This is illustrated in the figure 3.24a.
- **Forget Gate:** It takes the long-term memory LTM_{t-1} and it decides what parts to keep and to forget. It is multiplied by a **forget factor** f_t. This factor is again calculated using the short-term memory STM_{t-1}, the event and the sigmoid activation function. This is nothing more than a neural network using a linear function which gets activated by a sigmoid function. This is illustrated in the figure 3.24b.
- **Remember Gate:** The remember gate remembers the output coming from the long-term memory LTM_{t-1} and the short-term memory STM_{t-1}, combines (adds) them and creates a new long-term memory LTM_t. This is by adding the outputs from the learn gate and the forget gate as shown in the equation 3.39
- **Use Gate:** Use Gate This is the output gate which gives the prediction. The output is actually the new short-term memory STM_t. It is calculated as shown in the figure 3.24c and the equation

Finally, in the figure 3.24e, each LSTM takes in an input of a particular timestep. If the cell is considered as a blackbox, it is not that different from the RNN architecture.

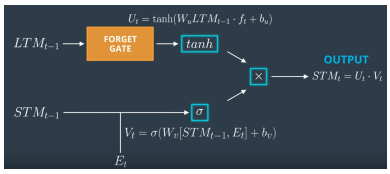
$$N_t = \tanh(W_n[STM_{t-1}, E_t] + b_n) \tag{3.39}$$
$$i_t = \sigma(W_i[STM_{t-1}, E_t] + b_i) \tag{3.40}$$
$$f_t = \sigma(W_f\, STM_{t-1}, E_t\ + b_f) \tag{3.41}$$
$$LTM_t = LTM_{t-1} f_t + N_t i_t \tag{3.42}$$
$$U_t = \tanh(W_u[LTM_{t-1} f_t] + b_u) \tag{3.43}$$
$$V_t = \sigma(W_v[STM_{t-1}, E_t] + b_v) \tag{3.44}$$



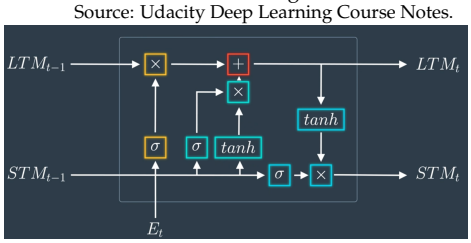
(a) LSTM-Learn Gate.
Source: Udacity Deep Learning Course Notes.



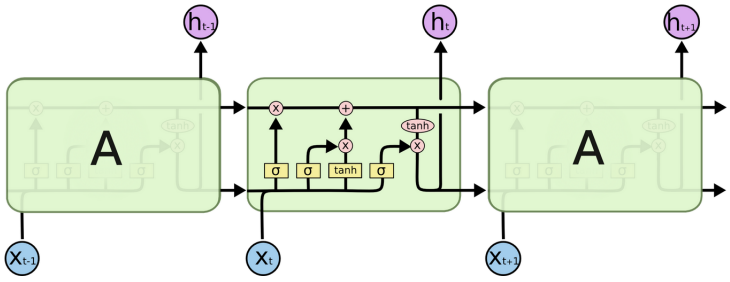
(b) LSTM-Forget Gate.
Source: Udacity Deep Learning Course Notes.



(c) LSTM Use Gate.
Source: Udacity Deep Learning Course Notes.



(d) LSTM Complete Architecture.
Source: Udacity Deep Learning Course Notes.



(e) LSTM Complete Architecture with Sequence.
Source: Understanding LSTM Networks

Figure 3.24: LSTM Complete Architecture.

3.9.5 Gated Recurrent Unit

It is one of many LSTM variations. Basically, it combines the forget and the learn gate into an update gate and then it is passed to another gate called the reset gate.

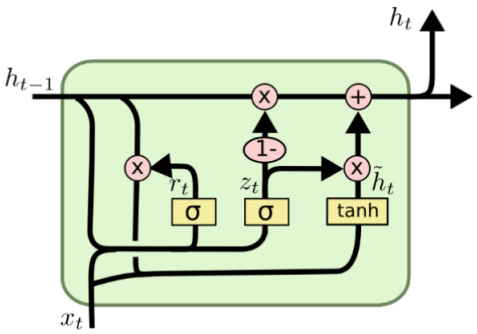
Unlike LSTM which creates a pair of long and short-term memory, GRU creates only one new working memory. The different gates in a GRU cell are:

- **Update Gate:** In GRU, this gate combines two gates from LSTM. The update gate decides which information to ignore and which to save and is denoted as z_t.
- **Reset Gate:** This gate decides how much of the information from the past to forget and is denoted as r_t

"These results clearly indicate the advantages of the gating units over the more traditional recurrentunits. Convergence is often faster, and the final solutions tend to be better. However, our results arenot conclusive in comparing the LSTM and the GRU, which suggests that the choice of the type ofgated recurrent unit may depend heavily on the dataset and corresponding task." [59]

Here are some applications examples created with GRU. Also, please take in account that the diagram illustrated in 3.23 is applicable for both LSTM and GRU.

- Speech Recognition - *Light Gates Recurrent Units for Speech Recognition* [60]. Interesting modified version of GRU, in the light version the $\tanh$ function is changed by a ReLU function. It turns out that this version performs 30% better in training time over a standard GRU.
- *Application of Gated Recurrent Unit (GRU) Network for Forecasting River Water Levels Affected by Tides* [61]
- *Applying deep learning approaches for network traffic prediction*[62]



$$z_t = \sigma(W_z[h_{t-1}, x_t]) \tag{3.45}$$
$$r_t = \sigma(W_r[h_{t-1}, x_t]) \tag{3.46}$$
$$\tilde{h}_t = \tanh(W[r_t * h_{t-1}, x_t]) \tag{3.47}$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{3.48}$$

Figure 3.25: Gated Recurrent Unit.
Source: GRU vs LSTM

3.10 Convolutional Neural Network

Convolutional Neural Networks (CNN) is vastly used for image feature extraction, the most important or known application being the image classification. However, CNN are not only famous for image processing, here are a list of well-known applications implemented with it. The very first model created that

- Text-to-Speech - *WaveNet: A generative model for raw audio* - Samples can be found Deep Mind.
- Text Classification - *Implementing a CNN for Text Classification in TensorFlow* - The code and the explanation can be found the official website: WILDML
- Language Translation - *A novel approach to neural machine translation* - Facebook Engineering.
- Save Endangered Species - *Reading Barcodes on Hooves: How Deep Learning Is Helping Save Endangered Zebras* - NVIDIA
- Speech Recognition - *Deep Speech 2: End-to-End Speech Recognition inEnglish and Mandarin* [63]. This application combines convolutional neural network with recurrent neural network and fully connected neural network.
- Many many more.

Convolutional Neural Networks are called so because a mathematical operation called **convolution** is used instead of a typical matrix multiplication and summation as seen until now.

The mathematical expression of convolution is as shown in the equation 3.49, in continuous and in discrete form. The convolution operation is represented by $*$ and with this calculation, one can know how the shape of function f is affecting the shape of g as shown in figure 3.26a.

Cross-correlation is different in a way that either f or g is flipped to know the correlation between those two functions. Actually, the convolutional neural networks operates the cross-correlation and not the convolution because the kernel is flipped whenever it has to feedfoward.

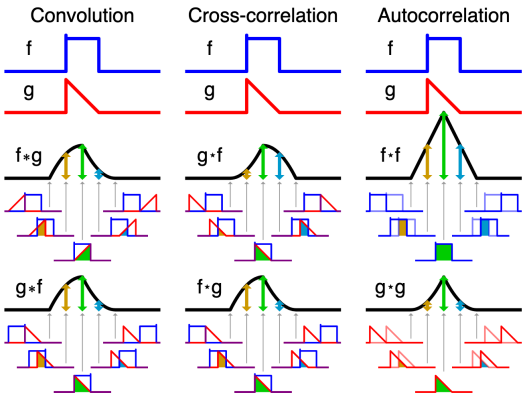
Autocorrelation is knowing the correlations of the function with itself.

Mapping the mathematical equation 3.49 to CNN, the function f would be the input of the network and the function g is known as the **kernel**. For example, if an image is the input of the network, the kernel will also have to be 2-D as the image and the convolution function would be something like as shown in the equation 3.26a [34]. In this equation, the image is I of dimension (m, n), the kernel K is also of the same dimension.

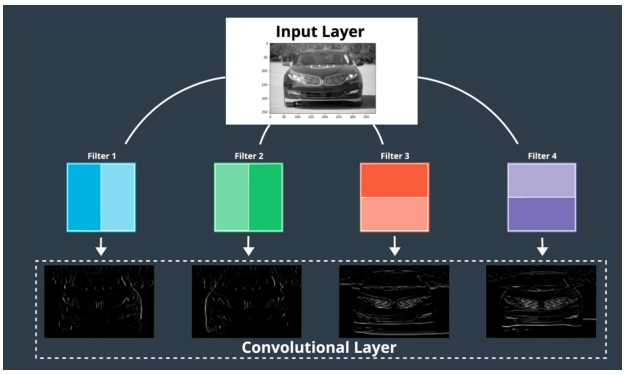
$$(f * g)(y) = \int f(\tau)g(t - \tau)d\tau \tag{3.49}$$

$$(f * g)[n] = \sum f[m]g[n - m] \tag{3.50}$$

$$S(i, j) = (I * K) = \sum\sum I(m, n)K(i - m, j - n) \tag{3.51}$$



(a) Convolution, Cross-Correlation and Autocorrelation
Source: By Cmglee - Own work, CC BY-SA 3.0



(b) Example of how filters are applied. In this case, to detect edges of the car.
Source: Udacity Deep Learning Course Notes.

3.10.1 Convolutional Layer

In Deep Learning, the kernel is also known as filter or the layer's weights. In the figure 3.27, the kernel moves horizontally and vertically through the image and an output is created. This is what is called the first layer of the CNN and is known as the **convolutional layer** and this is how the features are extracted of an image. The idea behind is that the kernel are initialized randomly just like in the neural networks and with training the kernel's should be optimized (the goal is to optimize the weights like before). Please note that the size of the output is determined by the size of the input as well as the size of the kernel.

Typically, ReLU activation function is applied after the convolutional layer. In the figure 3.26b, it can be seen how the filter can be useful for detecting edges.

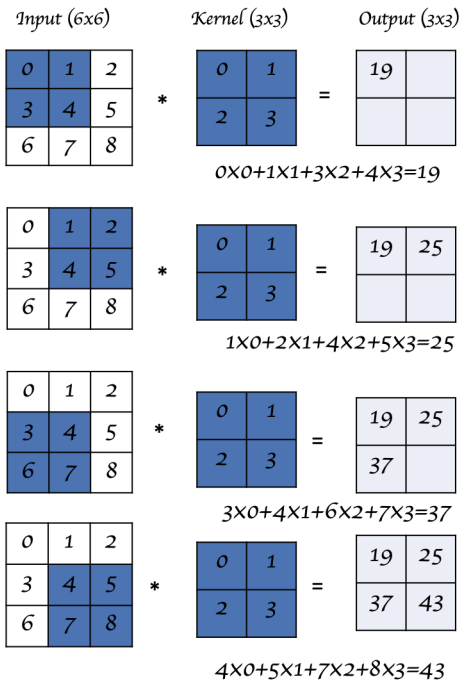


Figure 3.27: A simple demonstration of convolutional layer in a two dimensional input.

3.10.2 Padding

Looking again at the output of the previous step, it is clear that in the output, the pixel in the perimeters of the image are lost. If the network has layers and layers of convolution then this loss will add up and may harm the output. To avoid this problem, a technique called **padding** an be applied. In the figure 3.28, a padding of 0 is applied to the perimeters of the image. Take in account that only one layer of padding is applied. The value and the dimension of paddings is up the designer of the network.

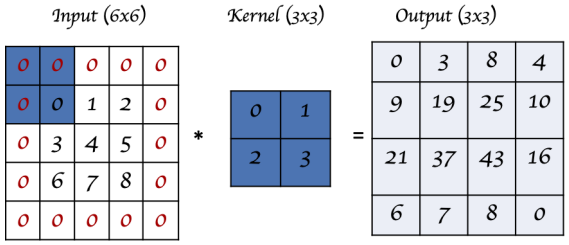


Figure 3.28: A demonstration of padding in a CNN with a two dimensional data.

The heights (columns) and the widths (rows) are denoted as h and w, respectively. So, if the shape of the input is $(n_h x n_w)$, the shape of the kernel is $(k_h x k_w)$, the shape of the padding is $(p_h x p_w)$ then the shape of the output will be $(n_h - k_h + p_h + 1)x(n_w - k_w + p_w + 1)$.

In the example, the input has the shape of $(3x3)$, the shape of the kernel is $(3, 3)$, the shape of the padding is $(3x3)$, the output has the shape of $(4, 4)$.

3.10.3 Stride

In the example seen above, the **stride** used was one. That is after applying the filter in the first corner of the image, the kernel moved one pixel vertically and then horizontally. If the stride is two, then the

kernel will move every two pixels as shown in the figure 3.29. Note that the perimeter with the dotted red boxes are not considered unless another layer of padding is added. Normally, stride is applied to improve computational efficiency or for downsampling.

The shape of the output is calculated of a convolutinal layer is calculated as $H = \frac{n_h - k_h + p_h}{s_h} + 1$ where H is the height of the output. $W = \frac{n_w - k_w + p_w}{s_w} + 1$

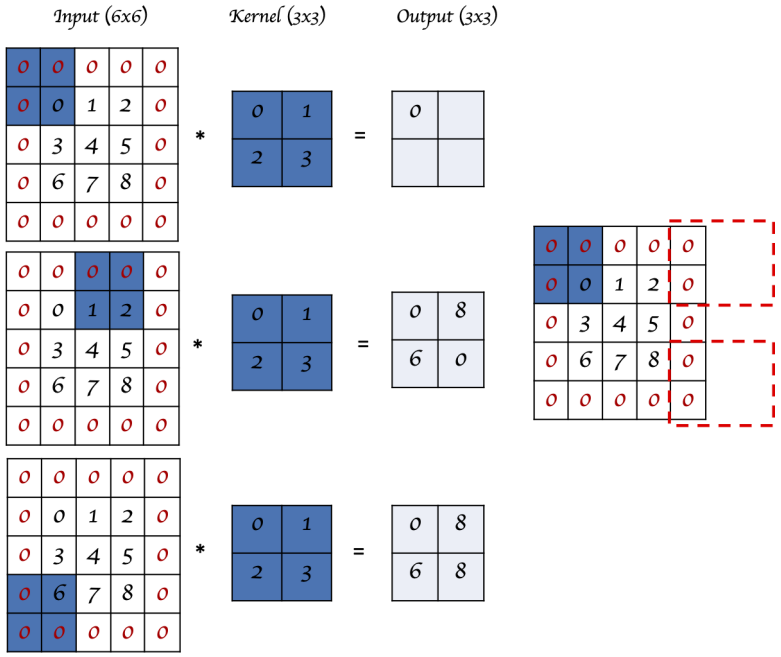


Figure 3.29: A demonstration of stride in a CNN with a two dimensional data.

3.10.4 Pooling

Two types of pooling, maximum pooling and average pooling. The pooling layers go after the convolutional layer and it depends on the designer to put this layer. According to *Ian Goodfellow et al.*, in the book Deep Learning [34], *pooling helps to make the representation approximately invariant to small translations of the input. Invariance to local translation can be a useful property if we care more about whether some feature is present than exactly where it is.* And then an example is given, for example, when an application is designed to know if the image contains a face, in that case, knowing where the eyes are localized is necessary.

An example as been shown in the figure 3.30. Th example shown is called max-pooling where on each window, the average value is calculated.

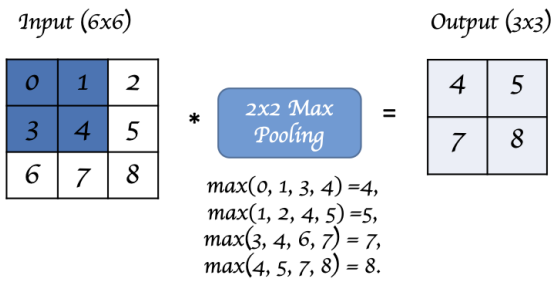


Figure 3.30: A demonstration of padding in a CNN in a two dimensional data.

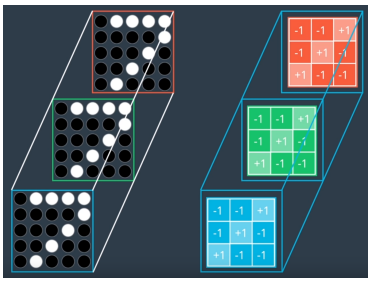
3.10.5 Multiple Dimension Input

Computers interpret the color images as three dimensional, height, width and depth. In case of RGB images, the depth or the channel is 3. It can be visualized as a stack of three two dimensional matrices as shown in the figure 3.31a. In this case, the filters are also multi-dimensional and each one computes on each dimension of the image.

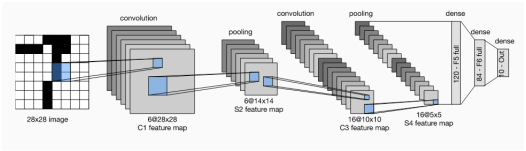
The figure 3.31b is the LeNet architecture[42]. It was experimented for recognizing handwritten digits (10 possible outputs). In this architecture, sigmoid function was used as activation function and not ReLU because this was not known until later. After each convolutional layer, there is a average-pooling layer and at the end of the feature extractions, there are two fully connected layer or also known as dense and then an output with 10 possible results.

This is actually a typical architecture of a convolutional network. A number of convolutional layers are stacked and then at least one fully connected layer is implemented to get the desired output. Another option is to connect to a recurrent neural network or both, to fully connected layer and then to the recurrent neural network. The possibilities of creating deep neural networks are limitless, although, there are guidelines and previous experiences, for now, the best results are obtained by trying and seeing the results.

In deep learning, for NN, to make things easier, instead of working in vectors and matrices, rather working in **tensors** is preferred. A tensor is a way of generalizing multi-dimensional vectors or matrices. For example, a vector $[1234]$ can also be called as a tensor of size 4. Examples are illustrated in figure 3.32.

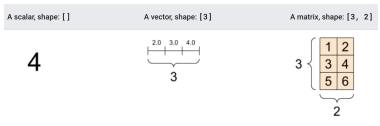


(a) A three dimensional image and a three dimensional kernel.

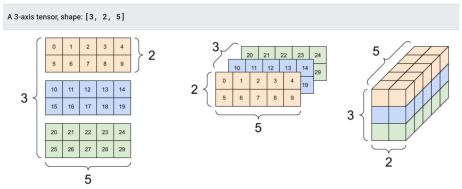


(b) Typical CNN Architecture.
Source: Dive into Deep Learning

Figure 3.31: Multi-dimensional Layer and LeNet Architecture.



(a) Difference between scalar, vector and tensor.



(b) Tensors in different format.

Figure 3.32: Example of Tensor.
Source: Tensorflow.

3.11 Deep Learning in Practice

The next chapter will put deep learning in practice to create an **Automatic Speech Recognition**. All the concepts learned in this chapter and as well as in the previous chapter will be applied and mentioned in the next one.

4

Experimenting with Neural Network Models

This chapter will explain the different types of models created, the reason of the experimentation and also the results.

The approach was to start with simple NN and then create more complex networks. Another important resource to take in account was the dataset.

4.1 Data

It was hard to find datasets in Spanish (without any economic cost), the dataset used for this project were these two with public domain license:

4.1.1 120h Spanish Speech Dataset:

120h of speech automatically extracted from Librivox. Credit to **Carlos Fonseca M**. This dataset has a total of 112,845 samples of audio, all read both by men and women with the corresponding label. An example can be seen in figure 4.1a. There are four columns:

1. **wav_filename:** The path of the audio *.wav* that will be the input to the model.
2. **wav_filesize:** The audio file's size.
3. **transcript:** This is the label or the utterance that is spoken in the audio file.
4. **duration:** The duration of the audio file in seconds.

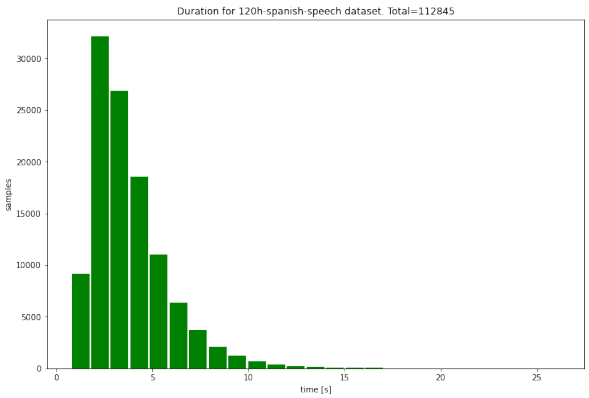
In the figure 4.1b, the distribution of the dataset is illustrated. Most of the files are between one to six seconds.

4.1.2 Spanish Single Speaker Speech Dataset

Librivox is again the source of this dataset with the peculiarity that only one man speaks in the whole dataset. There are 11,111 total audio files in this dataset with public domain license and credits to

	wav_filename	wav_filesize	transcript	duration
0	audios/4da6b70e-0108-4f75-80ae-3d71f1dd2c2b.wav	219064	y aquí en dos palotadas hemos encontrado robus...	6.824424
1	audios/8c2ab30b-0fd4-41c3-9724-3b15f2ee2c27.wav	271910	cuando los consejeros escucharon aquello queda...	8.470717
2	audios/ca73c951-c62a-41fe-a953-9871514151f2.wav	64520	su mujer con la cara entre las manos	2.009969
3	audios/067c4606-777b-4fb2-bc6f-8185fbec9016.wav	84222	y otros que se podían echar a la oreja de un toro	2.623738
4	audios/49a08f90-3fc0-43ad-bd5e-a6b671cafdd4.wav	77316	al oír mis pasos alzó la cabeza	2.408598

(a) Some rows of the dataset 120h-spanish-speech.
Source: From my $Jupyter$ Notebook $audio - data.ipynb$



(b) Data Distribution of 120h-spanish-speech: time vs samples.
Source: From my $Jupyter$ Notebook $audio - data.ipynb$

Figure 4.1: Look into 120h-speech-spanish Dataset.

Kyubyong Park. As shown in figure 4.2a, there are four columns:

- **filename:** The path to the audio file to read and process.
- **label:** The label or the utterance of what is spoken in the audio.
- **utterance:** Same as label, actually, no difference was found in these two columns. So, only one of them was used to create the label for the audio.
- **duration:** The duration of the audio in seconds.

In figure 4.2b, the duration of the audios is variable and most of them last between five to ten seconds. The audios in this dataset are longer than in the previous ones. Please note that both figures 4.1a 4.2a only are text files (*.csv* and *.txt*), these files only give the information about the names of the audiosfiles and the utterance of each of the files.

4.1.3 Data Pre-processing

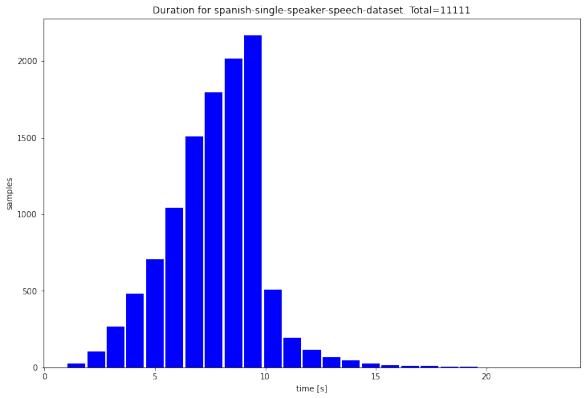
Before introducing the data to the model, the labels were cleaned up, the punctuation marks were replaced by empty character; the accents were replaced by the same characters without accents and finally the texts were converted to lowercase. The audios with no labels were discarded and also those audios with labels containing any digits were deleted because this would complicate the training process.

In all the models, the first dataset *Single Spanish Speaker* were used completed (except if some were discarded for reasons mentioned before) and the second one *120h Speech Spanish* was used only partially because using whole dataset meant consuming too much resources which was very limited during this project.

Not only does the model add computational complexity and inefficiency, the more data, the model has to process more which means the consumption time of the resources increases. Unless there is unlimited resources, the model and the data has to be balanced.

	filename	label	utterance	duration
0	19demarzo/19demarzo_0000.wav	Durante nuestra conversación advertí que la mu...	Durante nuestra conversación advertí que la mu...	5.88
1	19demarzo/19demarzo_0001.wav	Componíanla personas de ambos sexos y de todas...	Componíanla personas de ambos sexos y de todas...	4.52
2	19demarzo/19demarzo_0002.wav	espontáneamente venidas por uno de esos llamam...	espontáneamente venidas por uno de esos llamam...	8.31
3	19demarzo/19demarzo_0003.wav	y resuenan de improviso en los oídos de un pue...	y resuenan de improviso en los oídos de un pue...	7.20
4	19demarzo/19demarzo_0004.wav	La campana de ese arrebato glorioso no suena s...	La campana de ese arrebato glorioso no suena s...	7.50

(a) Single Spanish Speaker Dataset
Source: From my $Jupyter$ Notebook $audio - data.ipynb$



(b) Data Distribution of Single Spanish Speaker: time vs samples.
Source: From my $Jupyter$ Notebook $audio - data.ipynb$

Figure 4.2: A Look into Single Spanish Speaker Dataset.

4.1.3.1 Calculating the Spectrogram

As already seen in the section 2.1.4.3, in the previous chapter 2. The library librosa calculates the **mel-spectrogram** without too much of complication to the user.

Windows of $23[ms]$ were taken, which means that for each $23[ms]$, there are 128 features for time t. For example, an audio with $5.88[seconds]$ of duration, the calculations would be as follows, taking in account the explanation mentioned in the section 2.1.4.3.

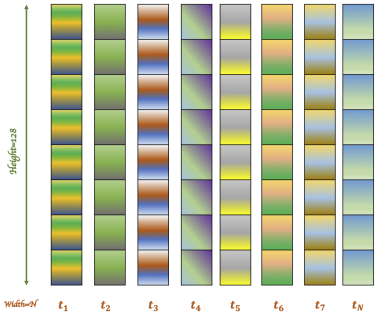
If the sample rate of the audio is $22050[\frac{samples}{second}]$ (librosa by default samples the audio to this value) and the window length (M in figure 2.10a) is $512[samples]$, the windows are of $23[ms]$. So, if the audio is of $5.88[seconds]$, the spectrogram will have $t = 255[frames]$

The mel-spectrogram calculated with the librosa library gives a vector of size $(128, t)$. With the caculations above, for the audio of $5.88[seconds]$, it would be $(128, 255)$. This is read as height of 128 mel bands or features at 255 time frames. So for each time frame there will be 255 frames. Muhammad Huzaifah [64] mentions that the Mel-Spectrograms with 512 or 128 features were the top performers among all the others. Also, in this project to reduce the dimension and computational cost, 128 was chosen and not 512. This can be seen in the figure 4.3a, where $N = 255$ in the example. Now, a matrix or a 2D vector or a 2D tensor of size $(H = 128, W = 255)$ has been created as is ready to be processed.

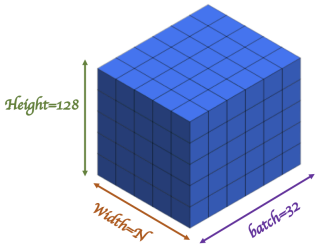
Only that, it is yet not ready to be processed, as mentioned in previous chapter, batches are needed for the training, one batch of size 32 should look like shown in figure 4.3b. However, the challenge was that the width was not fixed int datasets because each audio's duration is variable and now fixed and thus, making the width variable.

The solution to this was to pad the shorter ones. For example, is there are four vectors (for simplicity, put vector as example), $[1, 2, 3, 4]$ and $[5, 6, 7]$ and $[8, 9]$ then after padding the "pack" will become $[1, 2, 3, 4]$ and $[5, 6, 7, 0]$ and $[8, 9, 0, 0]$. Now, all have same sizes and the batch is whole. Please note that the batch is shown as 32 in the picture because for this project as for computational costs and

data size, it came to be the optimal one.



(a) Final 2D features-time matrix.



(b) 3D Illustration of a batch in Deep Learning

Figure 4.3: A Look into Single Spanish Speaker Dataset.

4.1.3.2 Char to Integer Mapping

One of the fundamental step in the data pre-processing is to convert the utterances to integers. This is because the output of the neural network will be numbers and the Softmax function will assign a probability of being a certain integer to each output for each time. So, for example, for a certain time, it will assign $[0, 1]$ probability to each of the integers. The output of the Softmax function is a probability distribution, not the desired output.

For example, in the figure 4.4, for each output of the neural network, looking at the output b, the Softmax assigns a probability of being one of the 29 integers (which in reality can be mapped back to char later).

After that, the results needs to go to a decoder, in this project, a **greedy decoder** has been implemented. This means that it will take the integer with the maximum probability (there are other types of decoders which takes in consideration the top 5, for example) and maps it back to the char using the map illustrated in the table 4.1.

char	""	" "	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
integer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

Table 4.1: Char-Integer Map

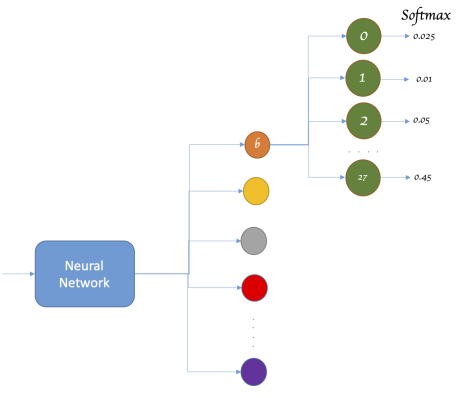


Figure 4.4: Softmax function's output is a probability distribution.

4.2 WER

How is the accuracy measured in a Speech-to-Text application? How is Siri's accuracy measured or Alexa's or any other paper designing ASR? It is measured by the **Word Error Rate**. It is $W = \frac{S+D+I}{N}$ where:

- **S is Substitution:** True word *casa* $\rightarrow$ predicted *cala*
- **D is Deletion:** True word *vamos a casa* $\rightarrow$ predicted *vamos _ casa*
- **I is Insertion:** True word *cala* $\rightarrow$ predicted *calla*
- **N:** is the total number of words spoken.

High WER means **less** accuracy and **Low** WER means **high** accuracy. The term WER comes from **Levenshtein distance**. The distance is equal to the number of changes that needs to be made to get the true word. For example, the Levenstein distance between casa and cala would be one because only one word has to be changed. This is so commonly used in Speech Recognition and Translation that there are snippets of codes in any programming language in Wikipedia.

4.3 GPU

GPU stands for Graphic Processing Unit. Initially, GPUs were meant for only gaming, specific for application with any graphical tasks. However, it started being flexible and now can be used for deep learning. On 2005, **Steinkraus et. al** [65] [34] experimented using GPUs for machine learning algorithms and the came to the following conclusion and definitely, their proposal was true.

We propose a generic 2-layer fully connected neural network GPU implementation which yields over 3x speedup for both training and testing with respect to a 3 GHz P4 CPU.

From then, on 2006 **Chellapilla et. al** [66] demostrated high performance CNN for document processing. For further knowledge of GPU, the chapter 12, Applications, from Deep Learning book[34] can be read, written by Ian GoodFellow.

To compare the computational costs between CPU and GPU, an example was ran. A simple neural network was been created using $pytorch$ as shown in the figure 4.5a with the following parameters:

- Two convolutional layers + One LSTM with 256 hidden layers + Fully connected layer.
- 12,000 audio files as inputs. 9,548 audio files as training dataset; 2,128 audio files as validation dataset and 237 audios as testing file.
- Batch size of 16.
- 1 epoch.

The results are as shown in 4.5, when ran in CPU and in GPU. There was a huge difference, in each epoch, training was done and as well as validation, the same process described and discussed in the previous chapter was followed.

- Figure 4.5b is the test done with CPU and it took about $14[minutes]$.
- Figure 4.5c is the test done with GPU and it took about $2[minutes]$. It ran 7 times faster than with CPU. Take in account that this test was don with a simple network. More complex networks means more parameters and more computational cost.

For this project, cloud environment kaggle was used. Kaggle is Google's subsidiary and it a community for data scientists and machine learning. People are allowed to publish datasets, create notebooks with those and publish them. There are often competitions held and the best thing about Kaggle is that they provide $30[hours]$ of GPU per week per account for free. There are two options to run the code, one is in the interactive session but if there is no activity in $60[minutes]$ then the session is stopped and also the progress will be lost. Another method for running the whole code is on the background, which also commits and saves the output, this option lets a user run a notebook for $9[hours]$ continuously. If the time exceeds, the progress is stopped and also the outputs are lost.

On 2019, Kaggle announced that Tesla P100 was being allowed to the users, so for this project this is one that was used.

The economic costs for GPU usage in different clouds for **NVIDIA Tesla P100** is as shown in the table 4.2. In the table **NVIDIA Tesla V100** is also included because Amazon Web Services doesn't offer the one used in this project. V100 was released one year after P100 and its performance is better than P100.

GPU	*Cloud*	*Price per GPU per hour*
NVIDIA® Tesla® P100	Google Cloud	1.46$
NVIDIA® Tesla® V100	Google Cloud	2.48$
NVIDIA® Tesla® V100	Amazon Web Services	3.06$
NVIDIA® Tesla® V100	IBM Cloud	3.06$
NVIDIA® Tesla® P100	IBM Cloud	1.95$
NVIDIA® Tesla® P100	Microsoft Azure	1.7457$
NVIDIA® Tesla® V100	Microsoft Azure	2.5805$

Table 4.2: Comparing GPU prices in different clouds.

It is worth mentioning that for testing purposes and for creating figures, another environment was also used. Google Colab is also a cloud environment and lets user create notebooks along with Google Drive. However, this environment wasn't used for training because the GPU provided here is X80 and its performance is not comparable to the one offered in Kaggle.

4.3.1 CUDA

CUDA stands for **Compute Unified Device Architecture**.It was developed by NVIDIA and are integrated in their GPU which makes it really easy while coding and taking full advantage of parallel computing.

It is worth mentioning **OpenCL**, a standard developed by Apple and Khronos. It is a generic standard and can be used in any GPU. Anyhow, no other has been able to beat NVIDIA in this area.

"CUDA is a parallel computing platform and programming model that makes using a GPU for general purpose computing simple and elegant" - Mark Ebersole.

```
ASRConvLSTM(
  (conv1): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv_layers): Sequential(
    (0): ConvLayer(
      (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      (ConvLayer): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (fully_connected): Linear(in_features=2048, out_features=512, bias=True)
  (lstm_layers): Sequential(
    (0): LSTMLayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(512, 256, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=256, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(a) A simple Neural Network Example in *pytorch*.

```
######################
#  Start Training    #
######################
Epoch 1: Training took 756.90 [s]       Validation took: 66.84 [s]

Epoch: 1 Losses Training Loss: 0.452054 Validation Loss: 0.445853
-------------------------------------------------------------------------
Epoch 1 took total 823.7393815517426 seconds
-------------------------------------------------------------------------
```

(b) CPU Run time for one epoch-simple network.

```
######################
#  Start Training    #
######################
Epoch 1: Training took 96.54 [s]        Validation took: 18.08 [s]

Epoch: 1 Losses Training Loss: 0.449003 Validation Loss: 0.417783
-------------------------------------------------------------------------
Epoch 1 took total 114.62229371070862 seconds
-------------------------------------------------------------------------
```

(c) GPU Run time for one epoch-simple network.

Figure 4.5: Comparing CPU with GPU.

4.4 Inspirations from Previous Works

This section reviews previous works on creating ASR using deep learning. Some of them were taken as inspiration and some of their conclusions lead to making easier decisions. These decisions were made to create efficient neural networks as the resources (like GPU) were limited in time, testing is a must in programming, and also please note that not only creating the ASR was investigated in this work. Even learning how to create neural networks using deep learning was a self-learning exercise to complete this project. Thus, consuming more of this limited resource.

4.4.1 Deep Speech

The figure 4.6a is from literature *Deep Speech: Scaling up end-to-endspeech recognition*[67], by Baidu Reasearch team. This networks' architecture and other details are as follows:

- Three fully connected layers to extract features at each time step.
- Then it is connected to a bi-directional RNN (nor LSTM nor GRU).
- It adds the result from both directions and applies it to another fully connected layer to transform the result.

- Finally, a softmax function is applied to calculate the probability at each time for each character.
- CTC Loss function was used.
- Dropout rate between 5% - 10%, only to the fully connected layers.
- ReLU as activation function.
- Audio data collection of about $5,000[hours]$ of read speech from $9,600$ speakers for training.
- To make it noise robust, training was done with 100 noisy and 100 noise-free uterrances from 10 speakers.

4.4.2 Deep Speech 2

The figure 4.6b is from literature *Deep Speech 2: End-to-End Speech Recognition in English and Mandarin* [63], by Baidu Research team. The purpose of this experiment was to prove that it is better than the previous one and the same architecture can recognize either English or Mandarin Chinese speech. 8 or 19 GPUs were used to train one model.

In this architecture, the authors explain that layers of recurrent layers doesn't have the capacity to train on thousands of hours of labelled speech so, they introduce depth (by convolutional layers). This model's architecture and other parameters are as follows:

- First layers consists of 1 to 2 convolutional layers.
- The number of recurrent or GRU layer varies from 1 to 7 layers. The hidden units of these RNN varies from 2,400 to 1,280 hiidden layers.
- The last layer consists of a fully connected layer which works as a classifier like in the previous architecture.
- Batch Normalization.
- It also uses the softmax function and the CTC loss function just like deep speech.
- Training was done with 11,490 hours of labeled speech containing 8 million in English.
- For Mandarin, mixed of read speech and spontaneous speech, in both standard Mandarin and accented Mandarin of 3,600 hours.

4.4.3 Convolutional Layer, Long Shot-Term Memory and Fully Connected Deep Neural Networks.

Th figure 4.6c is from the literature *Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks* [68], by Googl's team. This model is perhaps the one that approximates the most to the final model of this project because this type of network got the best results.

According to the authors, by adding the convolutional layer, the WER shows a relative improvement of 4-6%. The strategy is to perform on frequency dimension while on the convolutional layers and on time dimension while on the RNN. The architecture and other parameters of this model is as follows:

- Frames of x_t are injected as inputs, each timstep of 40-dimensional log-mel features.
- Two convolutional layers with max-pooling (only once).
- A linear layer (dim. red in the figure) to reduce the dimensions of the convolutional layers to outputs of 256. No loss in accuracy was observed.

- The next layer had 2 LSTM layers, where each layer has 832 cells.
- The number of last fully connected layers is not specified in the literature.
- Initially, the data used was about 200 hours in spoken English. Then more 2,000 hours audios (each of 3[minutes]) were added. These ones were atificially corrupted by adding noise.
- The trained model was evaluated in a clean test set containing 20 hours of audio.
- Additionally, another 30,000 uterrances with noise were also tested.

4.4.4 End-to-End Deep Neural Network for Automatic Speech Recognition

The figure 4.6d is from the literature *End-to-End Deep Neural Network for Automatic Speech Recognition*[69] by William Song and Jim Cai. This architecture is simple, it has 4 convolutional layers, each proceeded by a maxpooling layer and ReLU activation function. Followed by two dense layers, two RNN and then the CTC Loss function. The datast used in this model was from TMIT, it consists of 6,300 utterances in 8 dialects from 630 speakers.

4.4.5 Assembly AI Blog

Assembly AI is actually a speech recognition company and the most inspirational model for this project was actually from their blog: Building an end-to-end Speech Recognition model in PyTorch. Remarkable architecture, it actually surpasses any other architecture mentioned before. It is shown in figure 4.7. Instead of convolutional layers, it uses **Residual Networks**[70]. ACtually, this is another method of using deep layers of convolutional layer but in an efficient way.

4.4.6 Efficiency

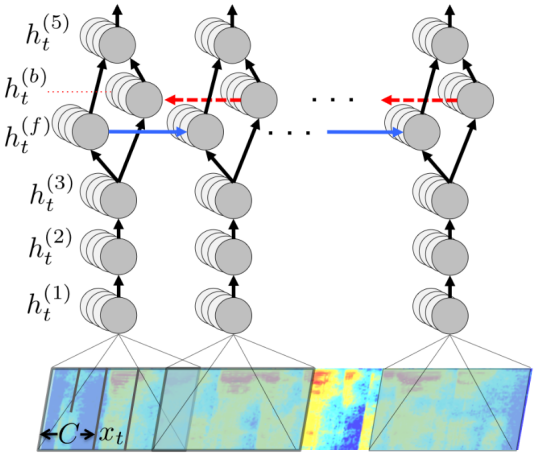
To make the most of the limited usage of free GPU, papers were read and investigated to find the optimal methods to reduces the time of trainings by reducing computational costs. The following were the methods used in this project:

4.4.7 Layer Normalization

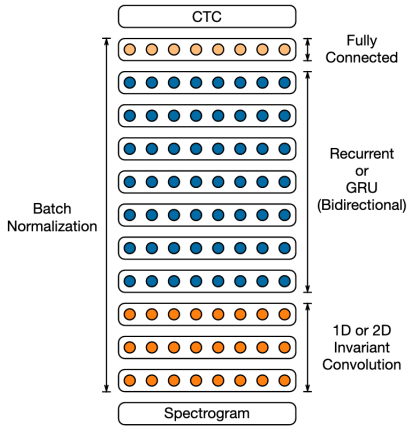
Batch Normalization [71], 2015, should be introduced before talking about Layer Normalization [72]. The conclusion of the literature is that at each layer of the neural network, the distribution of the data changes which makes the learning slower, the authors refer to this phenomenon as **internal covariate shift**. The idea of Batch Normalization is to normalize the data after each layer, the authors state that with this technique the weight initialization is not needed and even dropout could be forgotten.

The problem with batch normalization is that it normalizes over the batch, meaning that it normalizes for each and every channel of the convolutional network. This proposition of the author was specific to CNN and it was not clear if this could be used in RNN.

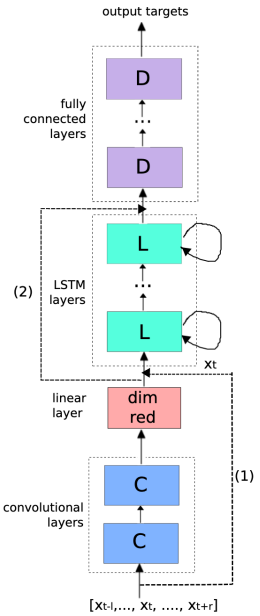
After a year on 2016, another type of normalization was proposed for RNN and it was called the **Layer Normalization** [72]. The problem with batch normalization for RNN was that the normalization would be applied to each time series separately and making it more complicated. As shown in figure 4.8, for recurrent networks, layer normalization works best because it normalizes the features for each x_t.



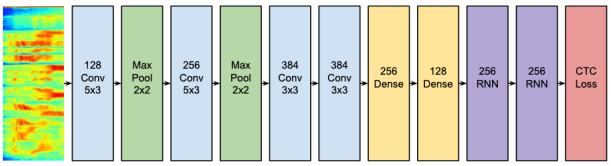
(a) Deep Speech Architecture [67].



(b) Deep Speech 2 Architecture [63].



(c) Convolutional Layer combined with RNN Architecture from Google Research [68]



(d) Deep Neural Network Architecture.[69]

Figure 4.6: Previous Works on Automatic Spech Recognition using Deep Learning.

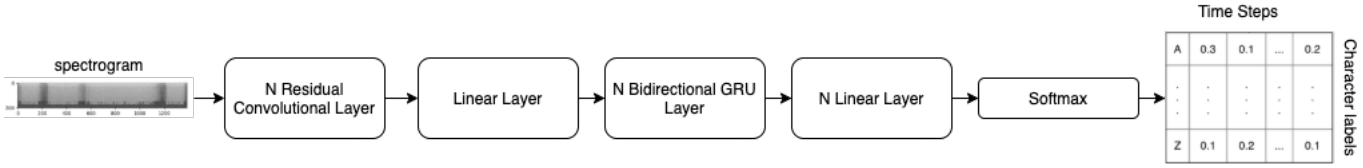


Figure 4.7: Assembly AI Architecture

4.4.8 Scaling the inputs

Scaling the inputs between $(-1, 1)$ was tried so that the distribution of the inputs were not sparse, however, this did not work. The patterns of the mel-spectrograms were probably lost by scaling these inputs, however the layer normalization method mentioned before, worked perfectly.

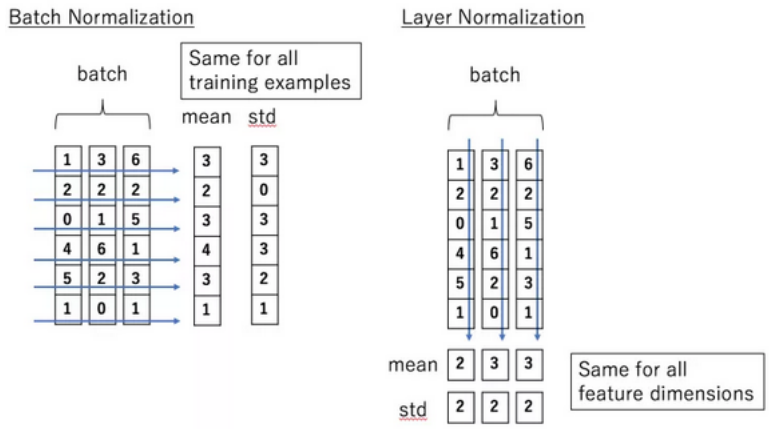


Figure 4.8: Batch Normalization works over batches and Layer Normalization works on features per each time.
Source: ML Explained.

4.5 Design and Implementation

The approach of the design was by reading papers with valid solutions and taking decision based upon those investigations. Methods were also used from previous investigations to improve efficiency. However, some of those efficiency were tested. For example, two models were tested to see if the layer normalization improved the computational cost. Both models had same parameters except for the layer normalization, one had it and the other didn't.

As the resource of the GPU was limited, testing was done before to run the whole code with only one epoch and check if everything worked as expected with CUDA, the settings, the models and for bugs. This was also very useful to calculate how many seconds it took for each epoch and depending on that, the parameters were configured. For example, if one epoch took like $13[minutes]$ then maximum epochs would have to be about 40 or another strategy would be to train with less samples or with less layers. The idea was always to find a balance between the different parameters in order to use the GPU int he most efficient way possible.

So, for all the models, the following pattern will be followed to explain the architectures, parameters, results and conclusions.

- Show and explain the architecture.
- The number of parameters that the model has and needs to adjust to get the desired output.
- Hyperparameters:
  - Number of epochs.
  - Hidden dimension was always 512 for all the RNN cells. Risk couldn't be taken to increase it because of computational costs. In all the literature mentioned in the previous sections, the hidden dimensions are massive, starting from about 1,024.
  - Batch size, almost always 32, sometimes 64 was tried but the RAM in kaggle ran out of memory it is limited to $13GB$.
  - Gradient Clipping, to avoid exploding gradients, clipping of 5 was used for all models. This is because in all models RNN are used.
  - Number of mel bands used was 64 for each time frame (frames of 10[ms] approximately).
  - Number of audio samples. Training, validation and testing.
- Hours of training time.
- Output examples.
- WER to measure the model.

4.5.1 Starting Simple

The first models were very simple ones, as the data to process is time-based, a RNN model was the first one that was obvious to test with. The models mentioned in this section can be found here: ASR-Spanish-Simple-RNN.ipynb and ASR-Spanish-Simple-RNN-lr-fixed.ipynb

The three models are very similar, all three of them share the following settings. The three models were executed once with a one cycle learning rate and once with a fixed learning rate of value 0.0001. All other parameters were the same in both execution except for the leaning rate.

- Number of epochs = 25
- Hidden dimension in all the GRU = 512
- Batch size = 32
- Training data size = 15,858 audios
- Validation data size = 3,568 audios
- Test data size = 397 audios
- Each layer has a dropout of 20% to avoid overfitting.
- The three of them have 4 GRU, with the ReLU as activation function. The layer is a fully connected layer which works as a classifier, gives out the probability of being one of the 28 chars from 4.1.

The model 1, as shown in figure 4.9a, and model 2, as shown in figure 4.10a, were implemented with 4 unidirectional GRU, while model 3, as shown in figure 4.11a, was implemented with bidirectional GRU.

Layer Normalization was not used in model 1, the difference of efficiency can be seen on table 4.3 between model 1 and model 2. Although, there is no difference in accuracy because both of them have the same WER rate. So, adding layer normalization to these only improved in training time but not in accuracy. However, this can only be confirmed if the models are trained for long hours.

The third model had both layer normalization and used bidirectional GRU. Even with few epochs, the accuracy rate was slightly lower than the first two, which means that bidirectional worked better for speech recognition because it used past, future and present information.

The plot shown on figure 4.12a compare the training and validation loss per epoch and per model. The black line, which is model 3's training loss seems to go down really fast compared to the other two models. However, the validation loss of this model 3 increases while the training loss doesn't stop decreasing. As seen in early chapter, this is a clear sign of overfitting. In this case, questions needs to be asked like, is the model to complex for the problem in hand? Is the dataset to small? Dropout technique is being used so, as in regularization that is fine.

In the same figure, the red line and the magenta line, from the first model, the simplest one to be going smooth, no drastic ups and downs like the second one and both the training loss (magenta) and the valid loss (red), seems to follow each other. So, *probably*, the third model is too complex for the problem in hand.

Some sample outputs are also shown from the test data and there was no huge difference between the three, it seems more than logical because the WER rate is similar in all cases.

4.5.1.1 Fixed Learning Rate

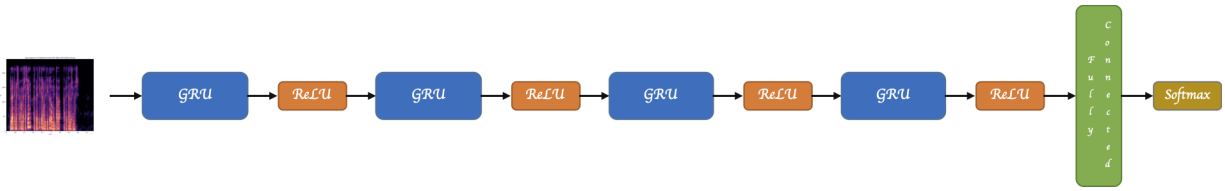
The outputs of the models with fixed learning rate of 0.0001 are better than the ones with a learning scheduler. The outputs from the fixed learning can be found in figure 4.22a. The third models is the one

with best outputs. The WER is worst in this model but because this model are forming more words than the one with a learning scheduler.

However, it is true that with a fixed learning rate the training times are higher but the results are better. The plots 4.12, compares the models with fixed and variable learning rates. The training and validation losses are smoother when the learning rate is fixed. In both cases, the model 3 (black line) is the one that starts overfitting.

Model	*Number of Parameters*	*WER*	*Total Time Training*
1	5,728,797	99.99%	1[hour]
2	5,731,869	96.99%	0.94[hour]
3	16,182,301	96.96%	1.68[hours]
1 with fixed lr	5,732,125	100%	1.175[hours]
2 with fixed lr	5,732,125	100%	1.14[hours]
3 with fixed lr	16,182,5571	100%	2[hours]

Table 4.3: Number of parameters, WER and total time training per model.



(a) First simple model - 4 layers of GRU with ReLU as activation function followed by a dropout=0.2 and finally a fully connected layer as a classifier.

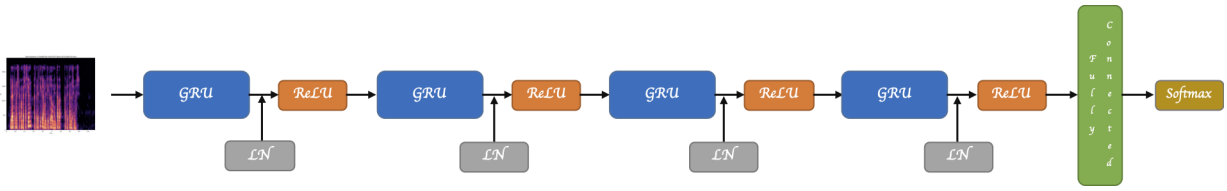
```
OnlyRNN(
  (gru_layers): Sequential(
    (0): GRULayer(
      (GRULayer): GRU(128, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): GRULayer(
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (2): GRULayer(
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (3): GRULayer(
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=512, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(b) Model 1 Object in pytorch.

```
######################
# Testing Model: Only-GRU #
######################


********************************************************************************
Predicted -- a ai
Utterance -- al recoger de mis manos la bandera el general despuesde preguntarme si era lice
nciado de presidio me dijo es vd sargento


********************************************************************************
Predicted -- ye e e e e a e e a e a a a a e a a a aea
Utterance -- nosotros apreciamos a este mala cabeza de santorcaz


********************************************************************************
Predicted -- ye e e e a e a a e e a e a e a a a e eea
Utterance -- imposibilitando de un modo absoluto la realizacion de mi proyecto
```

(c) Sample Output from Model 1.

Figure 4.9: Model 1 Architecture and Sample Output.



(a) Second simple model - 4 layers of GRU with layer normalization and ReLU as activation function followed by a dropout=0.2 and finally a fully connected layer as a classifier.

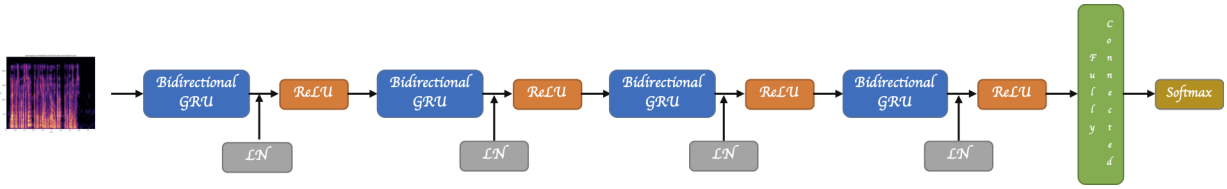
```
OnlyRNN(
  (gru_layers): Sequential(
    (0): GRULayer(
      (GRULayer): GRU(128, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (2): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (3): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=512, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(b) Model 2 Object in pytorch.

```
######################
# Testing Model: Only-GRU-Layer #
######################


********************************************************************************
Predicted -- ye e e e e
Utterance -- al recoger de mis manos la bandera el general despuesde preguntarme si era lice
nciado de presidio me dijo es vd sargento


********************************************************************************
Predicted -- e e e e e ee eea
Utterance -- nosotros apreciamos a este mala cabeza de santorcaz


********************************************************************************
Predicted -- ye e e e e e e ea
Utterance -- imposibilitando de un modo absoluto la realizacion de mi proyecto
```

(c) Sample Output from Model 2.

Figure 4.10: Model 2 Architecture and Sample Output.



(a) Third simple model - 4 layers of bidirectional GRU with layer normalization and ReLU as activation function followed by a dropout=0.2 and finally a fully connected layer as a classifier.

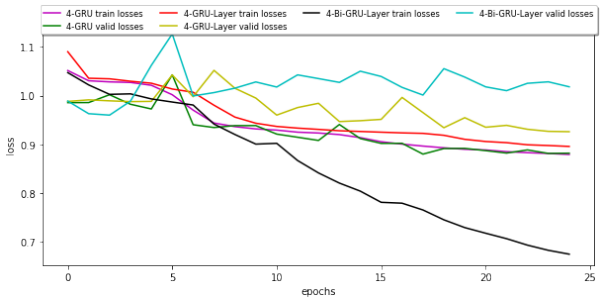
```
OnlyRNN(
  (gru_layers): Sequential(
    (0): GRULayer(
      (GRULayer): GRU(128, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (2): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (3): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=1024, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(b) Model 3 Object in pytorch.

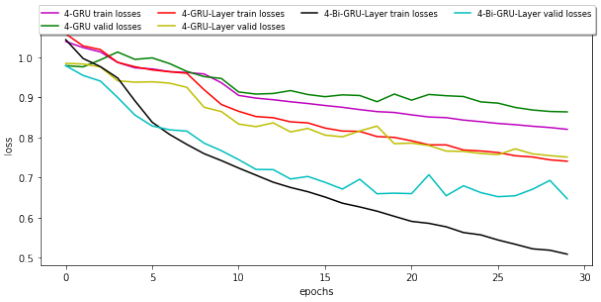
```
######################
# Testing Model: Only-GRU-Layer #
######################


********************************************************************************
Predicted -- d a
Utterance -- al recoger de mis manos la bandera el general despuesde preguntarme si era lice
nciado de presidio me dijo es vd sargento


********************************************************************************
Predicted -- sa a a araaa
Utterance -- nosotros apreciamos a este mala cabeza de santorcaz


********************************************************************************
Predicted -- y etrarararaera
Utterance -- imposibilitando de un modo absoluto la realizacion de mi proyecto
```

(c) Sample Output from Model 3.

Figure 4.11: Model 3 Architecture and Sample Output.



(a) Comparing the training loss and validation loss for three simple models.



(b) Comparing the training loss and validation loss for three simple models with fixed learning rate.

Figure 4.12: The three models comparison of training and validation losses.

```
######################
# Testing Model: Only-GRU #
######################


********************************************************************************
Predicted -- e ao
Utterance -- al recoger de mis manos la bandera el general despuesde preguntarme si era lice
nciado de presidio me dijo es vd sargento


********************************************************************************
Predicted -- ye e a e e eaeaos
Utterance -- nosotros apreciamos a este mala cabeza de santorcaz


********************************************************************************
Predicted -- e aeaos
Utterance -- imposibilitando de un modo absoluto la realizacion de mi proyecto


********************************************************************************
Predicted -- e aos
Utterance -- dijo que yendo de camino se habia extraviado no sabia donde estaba y buscaba al
bergue en que pasar la noche


********************************************************************************
Predicted -- e aos
Utterance -- a escocer a quemar de un modo muy vivo me decidi a leer la carta abiertaporque
el deseo de hacerlo
```

(a) Sample output for first simple model with a fixed learning rate.

```
######################
# Testing Model: Only-GRU-Layer #
######################


********************************************************************************
Predicted -- y ioesoa
Utterance -- al recoger de mis manos la bandera el general despuesde preguntarme si era lice
nciado de presidio me dijo es vd sargento


********************************************************************************
Predicted -- y e e e e a a as
Utterance -- nosotros apreciamos a este mala cabeza de santorcaz


********************************************************************************
Predicted -- y eoas
Utterance -- imposibilitando de un modo absoluto la realizacion de mi proyecto


********************************************************************************
Predicted -- y eoss
Utterance -- dijo que yendo de camino se habia extraviado no sabia donde estaba y buscaba al
bergue en que pasar la noche


********************************************************************************
Predicted -- y os
Utterance -- a escocer a quemar de un modo muy vivo me decidi a leer la carta abiertaporque
el deseo de hacerlo
```

(b) Sample output for second simple model with a fixed learning rate.

```
######################
# Testing Model: Only-GRU-Layer #
######################


********************************************************************************
Predicted -- acama ma ctoo
Utterance -- al recoger de mis manos la bandera el general despuesde preguntarme si era lice
nciado de presidio me dijo es vd sargento


********************************************************************************
Predicted -- noso cei a a e e a a a a e ataa
Utterance -- nosotros apreciamos a este mala cabeza de santorcaz


********************************************************************************
Predicted -- y sosiagando o o o o a a a a i i io
Utterance -- imposibilitando de un modo absoluto la realizacion de mi proyecto


********************************************************************************
Predicted -- y qeie d d d d i i a a a a a aces
Utterance -- dijo que yendo de camino se habia extraviado no sabia donde estaba y buscaba al
bergue en que pasar la noche


********************************************************************************
Predicted -- hxcumimimi i i i a a a a a e e cio
Utterance -- a escocer a quemar de un modo muy vivo me decidi a leer la carta abiertaporque
el deseo de hacerlo
```

(c) Sample output for third simple model with a fixed learning rate.

Figure 4.13: Sample outputs of the three models with fixed learning rate.

4.5.2 Convolutional Layers and LSTMs

This time, convolutional layers were added to extract the features before and then pass them to the RNN layers as seen in most of the previous works mentioned before. In this particular case, LSTM were used, the first one as shown in figure 4.14a were 4 unidirectional LSTMs.

The models mentioned in this section can be found here: ASR-Conv-LSTM.ipynb

Both models share the following parameters, the only difference is that one model uses unidirectional LSTM and the other bidirectional LSTM. In the table 4.4, there is no huge difference in execution time between both. There is a massive difference between the number of parameters, the model **Conv-Bi-LSTM** has about 24M parameters to adjust while the model **Conv-LSTM** has about 9M. The most surprising result is the **WER**, at first glance, it seems like the simple model performs better because the WER is less than in this case. However, looking at the figures 4.14c and 4.15c, the results are much better than the simple models. In these ones with convolutional layers, words start to form in the case of short utterances but in the long ones, seems like more training is needed. Then why is the WER higher in this case? If we look at the formula again $W = \frac{S+D+I}{N}$, in this case, there are substitute words to count as well as deletion and as well as insertion, while in the simpler model, only deletions were being counted.

The **first** convolutional layer has a kernel of size $(4, 4)$, stride of $(2, 2)$ and padding of $(1, 1)$. The input to this are the mel-spectrograms and they are injected in batches so they have size $(32, 1, 128, t)$, being t always variable as each audio have different duration. Channel is one because the spectrograms are grey scaled. This first convolutional layer separates the inout in 32 channels each (as shown in previous chapter) and extracts the features using the filter for each one of this channel. In every epoch and every batch, the network learns the feature comparing its output from the desird one (utterane). So, the output size will be $H = \frac{128-4+1}{2} + 1 = 64$ and $W = \frac{t-4+1}{2} + 1 = \frac{t}{2} = t'$.

The **second** convolutional layer has a kernel of size $(3, 3)$, stride of $(1, 1)$ and padding of $(1, 1)$. The output of this layer will be the same because the intention is to make the network learn the features in the same depth without futher modification, so $H = \frac{64-3+2}{1} + 1 = 64$ and $W = \frac{t'-3+2}{1} + 1 = t'$.

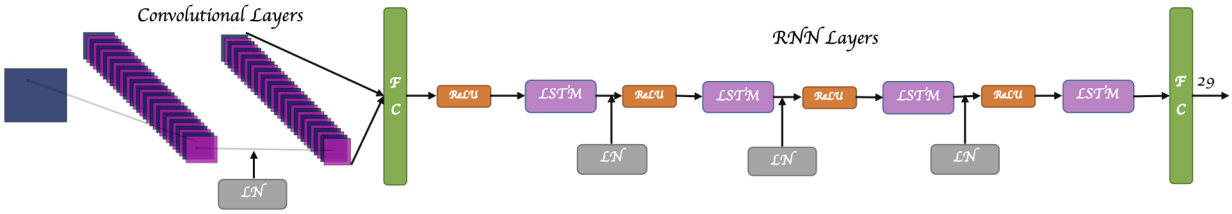
The fully connected layer after this layer, flattens the output so that there are less parameters to inject in the RNN layers .

The plot 4.16 shows the training loss and valid loss for both models. All the lines go down with a smooth slope. The training loss is slightly decreasing more than the validation loss in the case of bidirectional LSTM. Perhaps, the model needs much more data to train correctly and thus avoiding overfitting. Next, models created using GRU.

- Number of epochs = 25
- Hidden dimension in all the GRU = 512
- Batch size = 32
- Training data size = 15,858 audios
- Validation data size = 3,568 audios
- Test data size = 397 audios
- Each layer has a dropout of 20% to avoid overfitting.
- Both uses layer normalization for computational efficiency as this model is quite complex and has too many parameters.
- The three of them have 4 LSTM, with the ReLU as activation function. The layer is a fully connected layer which works as a classifier, gives out the probability of being one of the 28 chars from 4.1.

Model	*Number of Parameters*	*WER*	*Total Time Training*
Conv-LSTM	9,492,349	100%	1.56[hours]
Conv-Bi-LSTM	24,206,717	100%	2[hours]

Table 4.4: Number of parameters, WER and total time training per model.



(a) 2 Layers of Convolutional Layers, ReLU as activation function followed by 4 layers of LSTM with ReLU as activation function and finally a fully connected layer as a classifier.

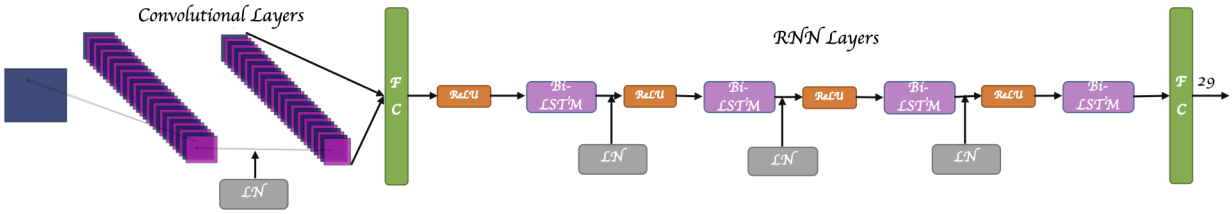
```
ASRConvLSTM(
  (conv1): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
  (fully_connected): Linear(in_features=2048, out_features=512, bias=True)
  (lstm_layers): Sequential(
    (0): LSTMLayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): LSTMLayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (2): LSTMLayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (3): LSTMLayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=512, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(b) Conv-LSTM Model Object in pytorch.

```
#####################
# Testing Model: Conv-LSTM #
#####################


********************************************************************************
Predicted -- ea sa a aosao
Utterance -- ya esta ya esta ahi albricias ya la tenemos ahi exclamo don carlos espana que a la
sazon de improviso se habia presentado


********************************************************************************
Predicted -- e eia e a e a e a aoeasa
Utterance -- en este dia la iglesia celebra ademas de la advocacion del carmen eltriunfo de la
santa cruz


********************************************************************************
Predicted -- perodoco de aa iioo ca cailo
Utterance -- pero cuando volvio de paris vino muy corregido


********************************************************************************
Predicted -- o e sini en eae esa oraes
Utterance -- porque los ingleses entraran en salamanca dijo santorcaz y no queremos que nos coj
an aqui


********************************************************************************
Predicted -- a a oe e o aoass
Utterance -- ahora con el trato se avivara el inmenso carino que os profesais desde hace alguno
s anos


********************************************************************************
Predicted -- poeso e deias
Utterance -- pues no te rias
```

(c) Sample Output from Conv-LSTM Model.

Figure 4.14: Conv-LSTM Model Architecture and Sample Output.



(a) 2 Layers of Convolutional Layers, ReLU as activation function followed by 4 layers of Bidirectional LSTM with ReLU as activation function and finally a fully connected layer as a classifier.

```
ASRConvLSTM(
  (conv1): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
  (fully_connected): Linear(in_features=2048, out_features=512, bias=True)
  (lstm_layers): Sequential(
    (0): LSTMLayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(512, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): LSTMLayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (2): LSTMLayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (3): LSTMLayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (LSTMLayer): LSTM(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=1024, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(b) Conv-Bidirectional-LSTM Model Object in pytorch.

```
######################
# Testing Model: Conv-BI-LSTM #
######################


********************************************************************************
Predicted -- ya a e e e e eososasae eaeaoao
Utterance -- ya esta ya esta ahi albricias ya la tenemos ahi exclamo don carlos es
pana que a la sazon de improviso se habia presentado


********************************************************************************
Predicted -- en e e e e e e araere
Utterance -- en este dia la iglesia celebra ademas de la advocacion del carmen elt
riunfo de la santa cruz


********************************************************************************
Predicted -- pero cado o do de de sio o o pa patio
Utterance -- pero cuando volvio de paris vino muy corregido


********************************************************************************
Predicted -- poes e e e an o oeno nocnini
Utterance -- porque los ingleses entraran en salamanca dijo santorcaz y no queremo
s que nos cojan aqui
```

(c) Sample Output from Conv-Bidirectional-LSTM Model.

Figure 4.15: Conv-Bidirectional-LSTM Model Architecture and Sample Output.

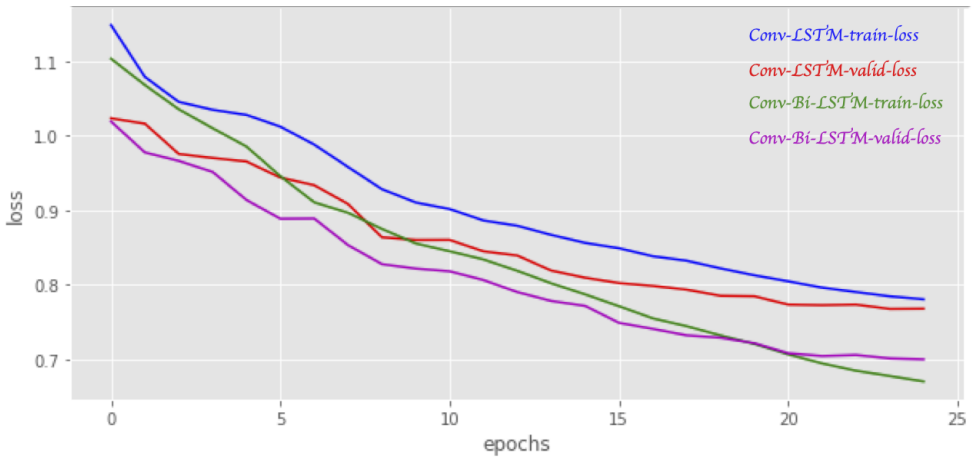


Figure 4.16: Comparing the training loss and validation loss for Conv-LSTM and Conv-BiLSTM models.

4.5.3 Model 2: Convolutional Layers with GRUs

This model is identical to the previous one with the exception that in this one, GRU cells are used. Two convolutional layers and four GRU layers as shown in figure 4.18 with unidirectional GRU and in the figure 4.18 with bidirectional GRU.

The models mentioned in this section can be found here: ASR-Conv-GRU.ipynb

In this case the training was done with 30 epochs while the previous ones were done with only 25. The training time with GRU is much less than the previous one as in GRU there are less gates than in LSTM.

In the table 4.5, the WER for the bidirectional gru is pretty good compared to the previous models and the unidirectional one. However, the number of parameters in the **Conv-Bi-GRU** is quite high, about $18M$ parameters and the unidirectional one has only about $7M$. Comparing these with the LSTMs, these are much lesser.

In the figure 4.18c, it is clear that the first words of the utterances starts to look very much as the utterance but in the end of the sentence, the outcome is not as expected. While, in the figure 4.17c, the words are forming but not as good as with the bidirectional ones.

In the plot 4.19, in both cases the lines descreases in a smooth manner. The bidirectional one decreases faster than the unidirectional ones, this behaviour is the same one spotter in the previous model as seen in figure 4.16.

- Number of epochs = 30
- Hidden dimension in all the GRU = 512
- Batch size = 32
- Training data size = 16,747 audios
- Validation data size = 3,768 audios
- Test data size = 419 audios
- Each layer has a dropout of 20% to avoid overfitting.
- Both uses layer normalization for computational efficiency as these models are quite complex and they have too many parameters.
- Both of them have 4 GRU, with the ReLU as activation function. The layer is a fully connected layer which works as a classifier, gives out the probability of being one of the 28 chars from 4.1.

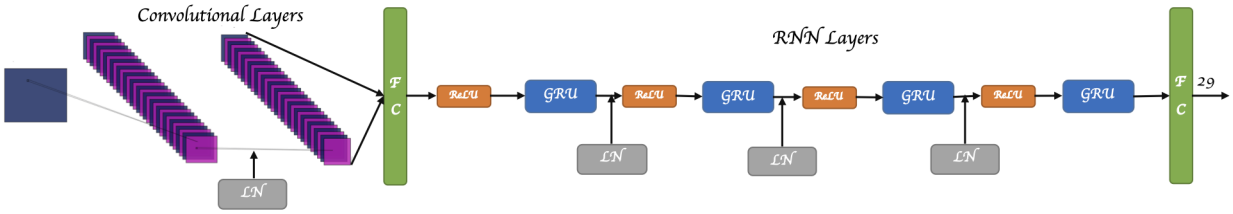
Model	*Number of Parameters*	*WER*	*Total Time Training*
Conv-GRU	7,381,725	100%	1.8[hours]
Conv-Bi-GRU	18,421981	100%	1.9[hours]

Table 4.5: Number of parameters, WER and total time training per model.

4.5.4 The Best Model

This model is identical to the one seen previously **Conv-Bi-GRU** 4.18a, nevertheless, this one has **six bidirectional GRU** layers after 2 convolutional layers (with the same parameters as before).

The models mentioned in this section can be found here: ASR-Best-Model.ipynb



(a) 2 Layers of Convolutional Layers, ReLU as activation function followed by 4 layers of GRU with ReLU as activation function and finally a fully connected layer as a classifier.

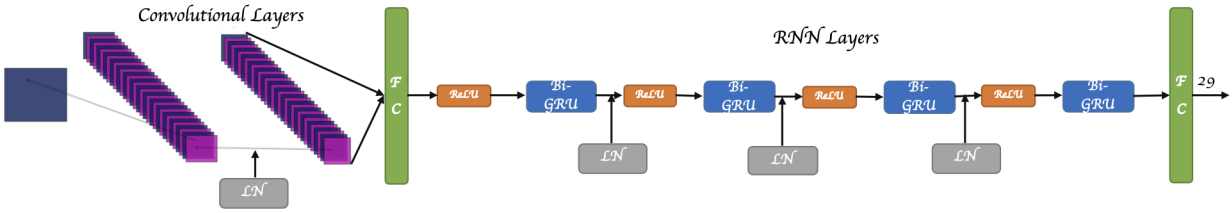
```
ASRConvBiGRU(
  (conv1): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv_layers): Sequential(
    (0): ConvLayer(
      (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      (ConvLayer): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (fully_connected): Linear(in_features=2048, out_features=512, bias=True)
  (gru_layers): Sequential(
    (0): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (2): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (3): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=512, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(b) Conv-GRU Model Object in pytorch.

```
######################
# Testing Model: Conv-GRU #
######################


********************************************************************************
Predicted -- ea a a a aeasoaosoaosatoasonodioisosalieresnado
Utterance -- ya esta ya esta ahi albricias ya la tenemos ahi exclamo don carlos espana que a la
sazon de improviso se habia presentado


********************************************************************************
Predicted -- eneste i e easaioeaoestonosanaos
Utterance -- en este dia la iglesia celebra ademas de la advocacion del carmen eltriunfo de la
santa cruz


********************************************************************************
Predicted -- perocado o io de pain dy o coido
Utterance -- pero cuando volvio de paris vino muy corregido


********************************************************************************
Predicted -- eor i oaonoaoreoseosonanati
Utterance -- porque los ingleses entraran en salamanca dijo santorcaz y no queremos que nos coj
an aqui


********************************************************************************
Predicted -- y a a a a asaosoaosoasasasosans
Utterance -- ahora con el trato se avivara el inmenso carino que os profesais desde hace alguno
s anos


********************************************************************************
Predicted -- por es o toe deias
Utterance -- pues no te rias
```

(c) Sample Output from Conv-GRU Model.

Figure 4.17: Conv-GRU Model Architecture and Sample Output.



(a) 2 Layers of Convolutional Layers, ReLU as activation function followed by 4 layers of Bidirectional GRU with ReLU as activation function and finally a fully connected layer as a classifier.

```
ASRConvBiGRU(
  (conv1): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv_layers): Sequential(
    (0): ConvLayer(
      (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      (ConvLayer): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (fully_connected): Linear(in_features=2048, out_features=512, bias=True)
  (gru_layers): Sequential(
    (0): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (2): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (3): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=1024, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(b) Conv-Bidirectional-GRU Model Object in pytorch.

```
######################
# Testing Model: Conv-BI-GRU #
######################


********************************************************************************
Predicted -- ya a as a a a a ae ao
Utterance -- ya esta ya esta ahi albricias ya la tenemos ahi exclamo don carlos espana que a la
sazon de improviso se habia presentado


********************************************************************************
Predicted -- en ea e e eoaeaorouero
Utterance -- en este dia la iglesia celebra ademas de la advocacion del carmen eltriunfo de la
santa cruz


********************************************************************************
Predicted -- pero pado oi eais io mo e catido
Utterance -- pero cuando volvio de paris vino muy corregido


********************************************************************************
Predicted -- pors e e e e a a a a e e eniqeinqiqiqiqi
Utterance -- porque los ingleses entraran en salamanca dijo santorcaz y no queremos que nos coj
an aqui


********************************************************************************
Predicted -- aba a a s saesososos
Utterance -- ahora con el trato se avivara el inmenso carino que os profesais desde hace alguno
s anos


********************************************************************************
Predicted -- poes o qe peias
Utterance -- pues no te rias
```

(c) Sample Output from Conv-Bidirectional-GRU Model.

Figure 4.18: Conv-Bidirectional-GRU Model Architecture and Sample Output.

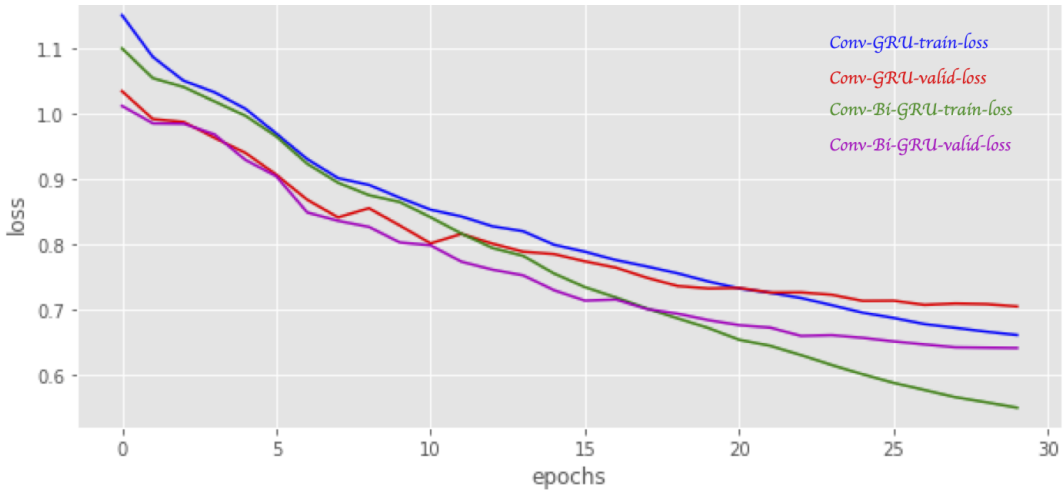


Figure 4.19: Comparing the training loss and validation loss for Conv-GRU and Conv-BiGRU models.

This model has almost $27M$ parameters but this is not much more than the model Conv-Bi-LSTM which had about $24M$ with only two bidirectional LSTM layers.

The final model was trained with almost $60K$ of audios, it presented 76.32% of WER, and was trained for $7[hours]$ as shown in the table 4.6.

In the figure 4.21c, the output of the model are quite good, and in the figure 4.21d, the short sentences predicted by the model are almost the same as the original utterances. However, for long sentences, the output is not that good which suggests that more training is needed, although, looking at the plot in figure 4.20, the model starts overfitting at about epoch 25. The previous model, Conv-BiGRU with 4 layers of GRU, starts overfitting before at about epoch 20 but this one starts later which suggests that adding more data can avoid overfitting.

This model can perform even better with more data and even tuning the parameters, the next chapter will show how this can be improved. The parameters of this model are:

- Number of epochs = 35
- Hidden dimension in all the GRU = 512
- Batch size = 32
- Training data size = 56,747 audios
- Validation data size = 12,768 audios
- Test data size = 1,419 audios
- Each layer has a dropout of 20% to avoid overfitting.
- Both uses layer normalization for computational efficiency as these models are quite complex and they have too many parameters.
- In this case there are two convolutional layers and textbfsix GRU, with the ReLU as activation function. The layer is a fully connected layer which works as a classifier, gives out the probability of being one of the 28 chars from 4.1.

Model	*Number of Parameters*	*WER*	*Total Time Training*
Best Model	27,884,925	76.32%	7[hours]

Table 4.6: Number of parameters, WER and total time training per model.

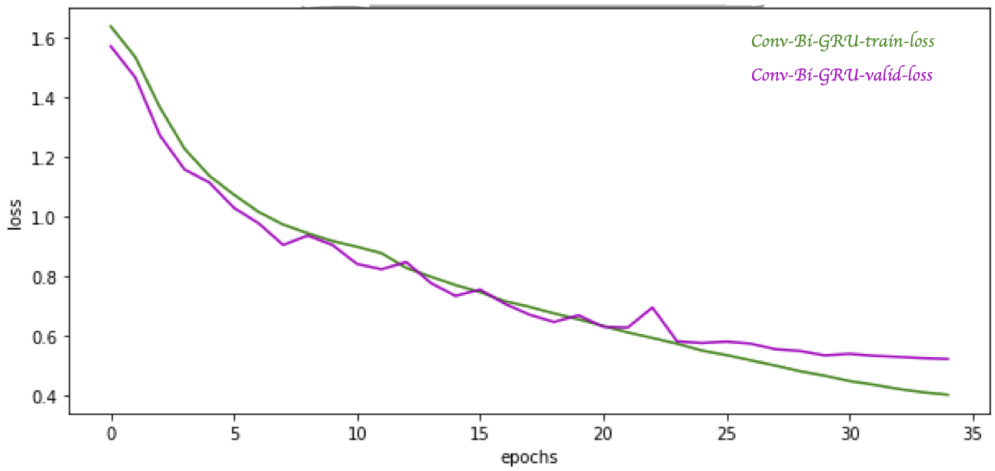
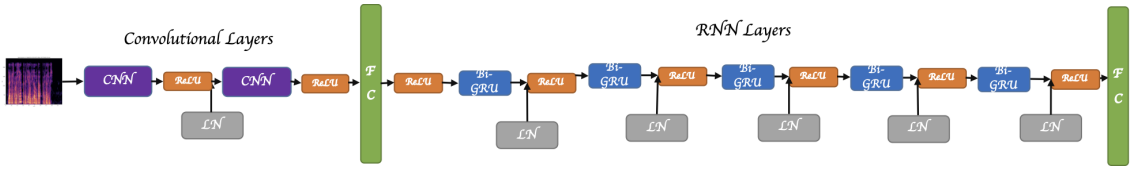


Figure 4.20: Comparing the training loss and validation loss for the best model.



(a) 2 Layers of Convolutional Layers, ReLU as activation function followed by 6 layers of bidirectional GRU with ReLU as activation function and finally a fully connected layer as a classifier.

```
ASRConvBiGRU(
  (conv1): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv_layers): Sequential(
    (0): ConvLayer(
      (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      (ConvLayer): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): ConvLayer(
      (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      (ConvLayer): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (fully_connected): Linear(in_features=2048, out_features=512, bias=True)
  (gru_layers): Sequential(
    (0): GRULayer(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(512, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (1): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (2): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (3): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (4): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (5): GRULayer(
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (GRULayer): GRU(1024, 512, batch_first=True, bidirectional=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
  )
  (classifier): Linear(in_features=1024, out_features=29, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

(b) 2 Convolutional Layers and 6 GRU Layers Model Object in pytorch.

```
######################
# Testing Model: Conv-BI-GRU #
######################


================================================================================
Predicted -- aleds despcaos de a que istne a dad osereseres
Utterance -- alento a sus esclavos de manera que en un instante llevaron al bergantin los
cofres y riquezas


================================================================================
Predicted -- el coz empezo abalparse
Utterance -- el juez empezo a palparse


================================================================================
Predicted -- no es mucho que des a tinme
Utterance -- no es mucho que desatine


================================================================================
Predicted -- sas e e a a a a e eaoiaae
Utterance -- estas palabras las oi confusamente y despues me quede solo o mejor dicho acom
panado de algunos chicuelos que me empujaban de aca para alla jugando conmigo


================================================================================
Predicted -- pues no ben aritio de venia gente
Utterance -- pues no bien advirtio que venia gente


================================================================================
Predicted -- de iso e a aeoa
Utterance -- bendito sea el senor omnipotente y misericordioso exclame despues de esto no
necesito mas que ojos y afortunadamente los tengo


================================================================================
Predicted -- tu que puedese entender
Utterance -- tu que puedes entender


================================================================================
Predicted -- juande dios comia en casa de lo re quejos
Utterance -- juan de dios comia en casa de los requejos


================================================================================
Predicted -- eso se a e o eaeaeaeaenenensaoaa
Utterance -- eso seria muy peligroso yo no puedo hacer eso sin ponerme antes de acuerdo co
n ella para que prepare su evasion con prudencia y sin escandalo
```

(c) Sample Outputs from the Best Model.

```
**********
Predicted -- no se le habia extenmido ilete
Utterance -- no se le habia extendido billete


**********
Predicted -- me queras toda via aque los tarlluces
Utterance -- me querras todavia a que buscar luces


**********
Predicted -- cati e ho piera tierra y me dijo que
Utterance -- cati echo pie a tierra y me dijo que


**********
Predicted -- dos podadores se harmaon con escalas
Utterance -- los pobladores se armaron con espadas


**********
Predicted -- cuando recidio est n esperada respuest
Utterance -- cuando recibio esta inesperada respuesta


**********
Predicted -- para hombrarmos con ella de dia
Utterance -- para honrarnos con ello de dia
```

(d) Short sentences are nearly the same as the real utterances.

Figure 4.21: The Best Model Architecture and Sample Outputs.

4.5.5 The best model with different learning rates.

In the previous model, one cycle learning rate was used (reduces chances of overfitting as explained in chapter 3). However, the same identical model explained in the section 4.5.4 was tested using a fixed learning rate and with another value of maximum learning rate for the one cycle (using the same as from the deep speech model).

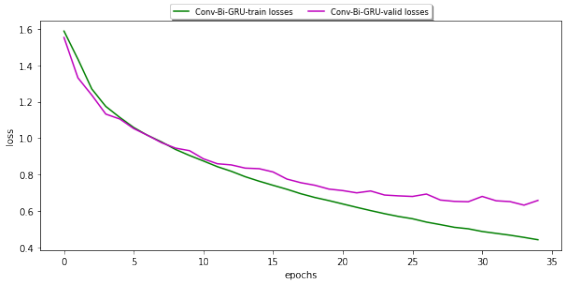
The models mentioned in this section can be found here: ASR-Conv-GRU-Fixed-LR.ipynb

The figure 4.22a shows some sample output for the best model with a fixed learning rate of value = 0.0001, the **WER** for this model was 81.74%. The figure 4.22c represents the training vs validation loss and actually the tendency is not that far away from the one in the figure 4.20, so actually, this model learns adequately at this rate **but**, with the fixed learning rate, overfitting starts happening at about epoch=10 while in the best model, overfitting happened much later.

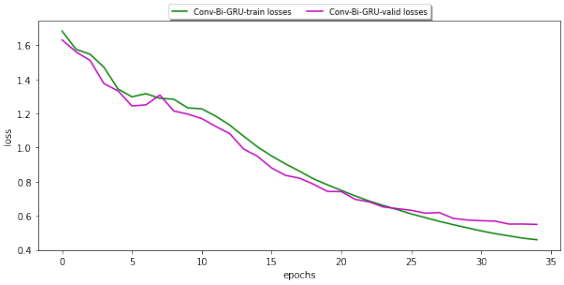
The figure 4.22b shows some sample output for the best model with a maximum learning rate from the model **Deep Speech** [67], the **WER** for this model was 81.45%, not far away from the fixed learning rate. The figue 4.22d shows that the learning rate is actually slow at first but later on it actually is very similar to the one of the best model shown in figure 4.20 and actually both starts with overfitting at about epoch=25.

(a) Sample output from the best Model with a fixed learning rate=0.0001

(b) Sample Output with learning rate as same as from the Deep Speech model.



(c) Training loss vs. Validation loss per epoch for fixed learning rate=0.0001



(d) Training loss vs. Validation loss per epoch for same learning rate as deep speech.

Figure 4.22: Outputs for the best model with alternative learning rates

4.5.6 Others

Two more models were tested with the same architecture as the one shown in figure 4.11a but one with 256 hidden dimension with fixed learning rate=0.001 and the other with 512 hidden dimension with fixed learning rate=10^-6.

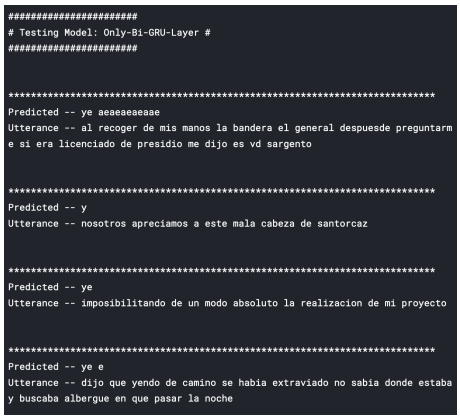
The models mentioned in this section can be found here: ASR-Simple-RNN-Others.ipynb

Plot shown in figure 4.23 shows that the model with 256 hidden dimension has a very high learning rate, thus it should be reduced because at some point, it stops learning. The one with 512 hidden dimension has a nice slop and the learning rate seems like is the adequate one for this model.

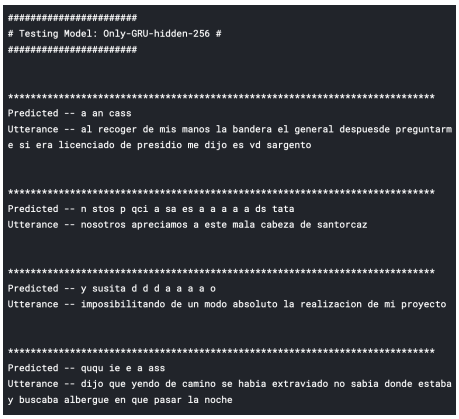
However, looking at the output samples of each as shown in figure 4.23b for the 256 hidden dimension and figure 4.23a for the 512 hidden dimension, the one with 256 outperforms the one with 512.

After trying these models with using convolutional layers, the question to answer would be if increasing the hidden dimension the actual reply to getting a better model or on the contrary, decreasing the hidden dimension?

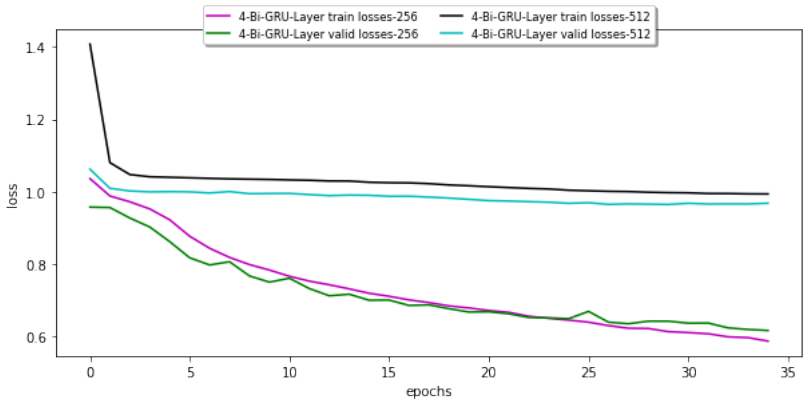
For the model with 256, there are $4,159,261$ parameters to adjust and for the one with 512 there are $16,182,557$. Obviously the second one will take longer to train but again, what about the overfitting?



(a) Sample output for the model as shown in figure 4.11a with fixed lr=10^-6.



(b) Sample output for the model as shown in figure 4.11a with fixed lr=0.001 and 256 hidden dimensions.



(c) Training loss vs. Validation loss per epoch for the alternative models.

Figure 4.23: Outputs for the best model with alternative learning rates and hidden dimensions

5

Conclusions and Improvements

The previous chapter demonstrated the theory learned in chapter 3 and why of the design and the implementation decisions. The experiments were done by comparing different models and by increasing the complexity. All these decisions were taken taking in mind the efficiency, the computational cost and lessons learned from past experiments. This chapter will focus on the conclusions and how can the models be improved.

5.1 Conclusions

The **first** lesson learned from the experiments was that audio data in Spanish was very difficult to find. The **second** one probably would be how costly training models can be, thanks to community like **kaggle** and **Google Colab** people can start experimenting deep learning.

The final result was satisfactory but it left many opens questions in the end like:

- Would it be worth trying with a less complex model? Would it solve the problem of overfitting?
- 128 number of mel bands were used, was this excessive? Choosing 64 may have been sufficient?
- What if the number of hidden layers in the RNN were less? May be the results would been the same with less computational cost?
- What would happen without the layer normalization?
- The learning rate was the most complicated hyperparameter to adjust and were taken as examples from other existing models.
- What if the final model was trained for more epochs? Would the outcome be precise also for the long sentences? This couldn't be answered as in kaggle, a model can be ran only for 9[hours] at once and to prove this, it would take probably more than 9[hours] and also to avoid overfitting, more data would be needed.

If the models experimented in the project are compared to the ones mentioned in the section where the inspirational models were previewed then the final model actually is very simple and needs much more improvement. In this inspirational models, the training times are measured in days, the GPU resources are unlimited as well as the data.

If one takes in account the limitations that were present in this project, it is clear that the results were more than expected, only trained for 7[hours] with only about $70K$ of audio data (average of 10[seconds]), and 76.32% WER.

The objective described in the first chapter was accomplished: *A Speech to Text was created using Deep Learning*. However, this trained model couldn't be tested with someone aged to know if all the objectives mentioned were accomplished. This could be one of the points to improve.

5.2 Improvements

Suggestions to improve the model are listed below:

1. Train with much more data, not only in volume but also in variety (different gender, age).
2. Injecting noise to the training dataset can significantly improve the training and it will be more robust to noise.
3. Train for long hours with sufficient data.
4. Try with more GRU layers or even more convolutional layers.
5. Adding more fully connected layers to eventually get to the last classifier layer would also have improved in performance.
6. Use **ResNet**[70] instead of convolutional layers.
7. Create a convolutional model with raw audio (without pre-processing) data would also have been an option as shown in the paper *End-to-End Speech Recognition from the Raw Waveform* [73].
8. Add **early stopping** to avoid overfitting. In this project, it wasn't used because the model didn't reach the optimal state as a speech-to-text.
9. Cloud based Machine Learning tools like Amazon SageMaker or Google Datalab could also be another option.
10. Create a website or a web server where people can upload their audio and produce text from the trained model.
11. Create a language model so that the words are assigned to the word most probable from a dictionary.

5.3 Closure

The output may not have been overall satisfactory, but the direction to follow is clear. Convolutional layers to extract features and RNN layers for the time-steps.

The most important lesson here to learn is that like in any computer science-related work, one needs to inspire or rely on previously existing work, to have a starting point, and then experiment and keep on with it. In deep learning, there are infinite possibilities, to tune the hyperparameters, get the exact, correct number, seems like a world of potential solutions but sometimes unreachable. Perhaps, because of lack of time or lack of resources, in terms of deep learning, or perhaps, whatever is in one's mind has still not been invented. Deep Learning is still on-progress science, many on-going theories in search of efficiency. After all, it is a science, and like any other science, people are desperate to find the only true optimal theory for this particular one, too. It would be marvelous to know the exact learning rate per type of network, the number of hidden layers for any purpose. But the search for a copy of a human brain in a machine can't be an exact science as there is still a lot to discover even about the human brain.

Acronyms

Adagrad Adaptive Gradients. 40

Adam Adaptive Moment Estimation. 40

AGI Aritifical General Intelligence. 23

AI Artificial Intelligence. v, ix, 1, 2, 4, 20, 21, 23–28

ANI Aritifical Narrow Intelligence. 23

ANN Aritifical Neural Network. 27

ASI Aritifical Super Intelligence. 23

ASR Automatic Speech Recognition. 3, 5, 10, 18, 20, 29, 61, 63

BERT Bidirectional Encoder Representations from Transformers. 33

BPTT Backpropagation Through Time. 46, 47

CNN Convolutional Neural Network. vi, 21, 51–55, 61, 65

CPU Central Processing Unit. 61

CTC Connectionist Temporal Classification. 38, 39

CUDA Compute Unified Device Architecture. 67

DARPA SUR Defense Advanced Research Projects Agency Speech Understanding Research. 21

DFT Discrete Fourier Transform. 18

DL Deep Learning. 2, 23, 25, 27, 29, 44

DNN Deep Neural Network. 30, 44

FFNN Feedforward Neural Network. 34, 45, 46, 49

FT Fourier Transform. 17, 18

GELU Gaussian Error Linear Unit. 32, 33

GPU Graphic Processing Unit. ix, 61–63, 65, 67, 83

GRU Gated Recurrent Unit. 50, 51, 64, 68, 72, 75, 78, 84

HMM Hidden Markov Model. 21

LSTM Long Short-Term Memory. 48–51, 65, 72, 75, 78

ML Machine Learning. 2, 23, 25, 26

MLP Multi-layer Perceptron. 30

MSE Mean Squared Error. 38

NN Neural Networks. 21, 29, 31, 34, 42, 55, 57

ReLU Rectified Linear Unit. 32, 51, 52, 55, 68, 72, 75, 78

RGB Red Green Blue. 55

RMSProp Root Mean Square Propagation. 40

RNN Recurrent Neural Network. vi, 21, 45–50, 63–68, 72, 83, 84

SGD Stochastic Gradient Descent. 38

STFT Short-Time Fourier Transform. 17, 18

TOH Threshold of Hearing. 11

WER Word Error Rate. 67–69, 75, 80, 84

Bibliography

[1] A.M Turing. *Computing Machinery and Intelligence*. 1950. URL: https://www.csee.umbc.edu/courses/471/papers/turing.pdf.

[2] Gordon E. Moore. *Cramming more components onto integrated circuits*. 1965. URL: https://newsroom.intel.com/wp-content/uploads/sites/11/2018/05/moores-law-electronics.pdf.

[3] The New York Times. *A.I. Is Learning to Read Mammograms*. 2020. URL: https://www.nytimes.com/2020/01/01/health/breast-cancer-mammogram-artificial-intelligence.html.

[4] Instituto Nacional de Estadíaticas de España. *Población residente en España. Provisionales a 1 de Julio de 2019*. 2019. URL: https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736176951&menu=ultiDatos&idp=1254735572981.

[5] Flanagan J.L. *Speech Analysis Synthesis and Perception*". Springer, Berlin, Heidelber, 1965. Chap. Acoustical Properties of the Vocal System.

[6] Vicente González Ruiz. *The Human Auditory System*. 2014. URL: https://w3.ual.es/~vruiz/Docencia/Apuntes/Perception/Sistema_Auditivo/index.html.

[7] The University of Texas and Health Science Center of Houston. *NeuroScience Online*. 2020. URL: https://nba.uth.tmc.edu/neuroscience/m/s2/index.htm.

[8] Pete Warden. *Launching the Speech Commands Dataset*. 2017. URL: https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html.

[9] Xuedon Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing. Huang, Xuedong and Acero, Alex and Hon, Hsiao-Wuen*. Prentice Hall, Jan. 2001.

[10] Kevin Wyffels et al. "Development of a Ground Truth Simulator and Application of a Generalized Multiple-Model Adaptive Estimation Approach to Tune a State Estimation Filter". In: (June 2020), p. 20.

[11] IBM. *Pioneering Speech Recognition*. URL: https://www.ibm.com/ibm/history/ibm100/us/en/icons/speechreco/.

[12] Wikipedia. *Speech Recognition*. URL: https://en.wikipedia.org/wiki/Speech_recognition#cite_note-NIPS2009-40.

[13] Lawrence R. Rabiner. "Automatic Speech Recognition - A Brief History of the Technology Development". In: 2004. URL: https://www.semanticscholar.org/paper/Automatic-Speech-Recognition-A-Brief-History-of-the-Rabiner/1d199099a2f4f8749c7e10480b29f5ada

[14] L. R. Rabiner and B. H. Juang. "An introduction to hidden Markov models". In: *IEEE ASSp Magazine* (1986). URL: http://ai.stanford.edu/~pabbeel/depth_qual/Rabiner_Juang_hmms.pdf.

[15] Xuedong Huang et al. "The SPHINX-II Speech Recognition System:An Overview". In: (1992). URL: https://pdfs.semanticscholar.org/9269/fbc18214e47da663ea04ed23c1cd396e0c38.pdf.

[16] Robert Strong. "Casper: A Speech Interface For The Macintosh". In: (1993). URL: https://www.isca-speech.org/archive/archive_papers/eurospeech_1993/e93_2073.pdf.

[17] Google. *About GOOG-411*. URL: https://web.archive.org/web/20071014091650/https://www.google.com/goog411/.

[18] Herve Bourlard and Nelson Morgan. *Connectionist Speech Recognition: A Hybrid Approach*. Jan. 1994. DOI: 10.1007/978-1-4615-3210-1. URL: https://www.researchgate.net/publication/230875873_Connectionist_Speech_Recognition_A_Hybrid_Approach.

[19] Alex Graves et al. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 369–376. ISBN: 1595933832. DOI: 10.1145/1143844.1143891. URL: https://doi.org/10.1145/1143844.1143891.

[20] Ava Mutchler. *Voice Assistant Timeline: A Short History of the Voice Revolution*. 2017. URL: https://voicebot.ai/2017/07/14/timeline-voice-assistants-short-history-voice-revolution/.

[21] François Chollet. *Deep Learning with Python*. Manning, Nov. 2017. ISBN: 9781617294433.

[22] Forbes. *Everything You Need To Know About Sophia, The World's First Robot Citizen*. 2017. URL: https://www.forbes.com/sites/zarastone/2017/11/07/everything-you-need-to-know-about-sophia-the-worlds-first-robot-citizen/.

[23] The Verge. *Sophia the robot's co-creator says the bot may not be true AI, but it is a work of art*. 2017. URL: https://www.theverge.com/2017/11/10/16617092/sophia-the-robot-citizen-ai-hanson-robotics-ben-goertzel.

[24] David Premack and Guy Woodruff. "Does the chimpanzee have a theory of mind?" In: *Behavioral and Brain Sciences* 1.4 (1978), pp. 515–526. DOI: 10.1017/S0140525X00076512.

[25] Shelley Duval. *A theory of objective self awareness.* 1972. URL: https://psycnet.apa.org/record/1973-26817-000.

[26] Interesting Engineering. *The Three Types of Artificial Intelligence: Understanding AI*. 2019. URL: https://interestingengineering.com/the-three-types-of-artificial-intelligence-understanding-ai.

[27] Forbes. *7 Types Of Artificial Intelligence*. 2019. URL: https://www.forbes.com/sites/cognitiveworld/2019/06/19/7-types-of-artificial-intelligence/.

[28] Government Technology. *Understanding the Four Types of Artificial Intelligence*. 2016. URL: https://www.govtech.com/computing/Understanding-the-Four-Types-of-Artificial-Intelligence.html.

[29] H. Chan et al. "Advancing Drug Discovery via Artificial Intelligence". In: *Trends in Pharmacological Sciences* 40 (July 2019). DOI: 10.1016/j.tips.2019.06.004.

[30] Stockholms Universitet. *Wolfgang von Kempelen's speaking machine and its successors*. URL: https://www2.ling.su.se/staff/hartmut/kemplne.htm.

[31] Jason Brownlee PhD. *14 Different Types of Learning in Machine Learning*. URL: https://machinelearningmastercom/types-of-learning-in-machine-learning/.

[32] MathWorks from Matlab. *Sophia the robot's co-Machine Learning. Tres cosas que es necesario saber*. URL: https://es.mathworks.com/discovery/machine-learning.html.

[33] Benedict Bikombo. *THE ULTIMATE GUIDE TO MACHINE LEARNING*. URL: https://aio.cloudaccess.host/machine-learning/.

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[36] Dan Hendrycks and Kevin Gimpel. "Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units". In: *CoRR* abs/1606.08415 (2016). arXiv: 1606.08415. URL: http://arxiv.org/abs/1606.08415.

[37] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[38] John C. Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." In: (2011). URL: https://stanford.edu/~jduchi/projects/DuchiHaSi10_colt.pdf.

[39] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. *Lecture 6a: Overview of mini-batch gradient descent.* URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[40] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[41] Leslie N. Smith. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. 2018. arXiv: 1803.09820 [cs.LG]. URL: https://arxiv.org/pdf/1803.09820.pdf.

[42] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *CoRR* abs/1206.5533 (2012). arXiv: 1206.5533. URL: http://arxiv.org/abs/1206.5533.

[43] Andrej Karpathy. *CS231n Convolutional Neural Networks for Visual Recgnition*. URL: https://cs231n.github.io/neural-networks-1/.

[44] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958.

[45] Jeffrey L. Elman. *Cognitive Science*. 1990. Chap. Finding Structure of Time, pp. 179–211. URL: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1.

[46] Wei Bao, Jun Yue, and Yulei Rao. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory". In: *PLoS ONE* 12 (July 2017). DOI: 10.1371/journal.pone.0180944.

[47] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 1310–1318. URL: http://proceedings.mlr.press/v28/pascanu13.html.

[48] Alex Graves and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *NEURAL NETWORKS* (2005), pp. 5–6.

[49] Alex Graves, Navdeep Jaitly, and Abdel rahman Mohamed. "Hybrid speech recognition with Deep Bidirectional LSTM". In: *ASRU*. 2013.

[50] Keras Documentation. *Trains a Bidirectional LSTM on the IMDB sentiment classification task*. URL: https://keras.io/examples/imdb_bidirectional_lstm/.

[51] Karthik Veerakumar. *Review generation using Bidirectional Long Short Term Memory(LSTM)*. URL: https://mc.ai/review-generation-using-bidirectional-long-short-term-memorylstm/.

[52] Abhyuday N Jagannatha and Hong Yu. "Bidirectional RNN for Medical Event Detection in Electronic Health Records". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 473–482. DOI: 10.18653/v1/N16-1056. URL: https://www.aclweb.org/anthology/N16-1056.

[53] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[54] Hermann Mayer et al. *A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks*.

[55] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. "Speech Recognition with Deep Recurrent Neural Networks". In: *CoRR* abs/1303.5778 (2013). arXiv: 1303.5778. URL: http://arxiv.org/abs/1303.5778.

[56] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. "An Application of Recurrent Neural Networks to Discriminative Keyword Spotting". In: *Artificial Neural Networks – ICANN 2007*. Ed. by Joaquim Marques de Sá et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 220–229. ISBN: 978-3-540-74695-9.

[57] Douglas Eck and Jürgen Schmidhuber. "Learning the long-term structure of the blues". In: *IN PROC. INTL. CONF*. 2002, pp. 284–289.

[58] Alex Graves et al. "Unconstrained Online Handwriting Recognition with Recurrent Neural Networks". In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS'07. Vancouver, British Columbia, Canada: Curran Associates Inc., 2007, pp. 577–584. ISBN: 9781605603520.

[59] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". English (US). In: *NIPS 2014 Workshop on Deep Learning, December 2014*. 2014.

[60] Mirco Ravanelli et al. "Light Gated Recurrent Units for Speech Recognition". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.2 (2018), pp. 92–102. ISSN: 2471-285X. DOI: 10.1109/tetci.2017.2762739. URL: http://dx.doi.org/10.1109/TETCI.2017.2762739.

[61] Xuan Hien Le, Hung Viet, and Giha Lee. "Application of Gated Recurrent Unit (GRU) Network for Forecasting River Water Levels Affected by Tides". In: Jan. 2020, pp. 673–680.

[62] R. Vinayakumar, K. P. Soman, and P. Poornachandran. "Applying deep learning approaches for network traffic prediction". In: (2017), pp. 2353–2358.

[63] Dario Amodei et al. "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin". In: *CoRR* abs/1512.02595 (2015). arXiv: 1512.02595. URL: http://arxiv.org/abs/1512.02595.

[64] Muhammad Huzaifah. "Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks". In: *CoRR* abs/1706.07156 (2017). arXiv: 1706.07156. URL: http://arxiv.org/abs/1706.07156.

[65] D. Steinkraus, I. Buck, and Patrice Simard. "Using GPUs for machine learning algorithms". In: Oct. 2005, 1115–1120 Vol. 2. DOI: 10.1109/ICDAR.2005.251.

[66] Kumar Chellapilla, Sidd Puri, and Patrice Simard. *High Performance Convolutional Neural Networks for Document Processing*. 2006. URL: https://hal.inria.fr/inria-00112631/document.

[67] Awni Y. Hannun et al. "Deep Speech: Scaling up end-to-end speech recognition". In: *ArXiv* abs/1412.5567 (2014).

[68] T. N. Sainath et al. "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks". In: (). URL: https://static.googleusercontent.com/media/research.google.com/es//pubs/archive/43455.pdf.

[69] William Song and Jim Cai. "End-to-End Deep Neural Network for AutomaticSpeech Recognition". In: (). URL: https://cs224d.stanford.edu/reports/SongWilliam.pdf.

[70] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/pdf/1512.03385.pdf.

[71] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[72] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[73] Neil Zeghidour et al. "End-to-End Speech Recognition from the Raw Waveform". In: *Interspeech 2018* (2018). DOI: 10.21437/interspeech.2018-2414. URL: http://dx.doi.org/10.21437/Interspeech.2018-2414.