Alma Mater Studiorum · University of Bologna

School of Economics, Management, and Statistics

Master of Science in Quantitative Finance

Natural Language Processing for Trading Signals using Deep Learning

Author:
Mattia Vicari

Supervisor:
Prof. Mauro Gaspari

III Session
Academic Year 2018/2019

Abstract

This thesis aims at investigating if and to what point it is possible to trade on sentiment and if Artificial Intelligence (AI) would be the best way to do so.

Despite the strict bindings of the Efficient Market Hypothesis (EMH) it appears that some kind of trading opportunity might be available. It may not be a case that many of those so-called "anomalies" appear to be, in reality, regular irrational behaviors done by financial agents. In fact, behavioral finance rose a wide literature and multiple debates supporting this statement. For this reason, I indicate an alternative to the EMH called Adaptive Market Hypothesis (AMH) whose benefit is incorporating the best of both worlds.

With this more appropriate theoretical basis we will take inspiration from previous works, such as Sohangir 2018 [29] or Ruiz-Martínez 2012 [19], where they show that it is possible to extract and use the market sentiment, specifically news, through computational methods in the world dominated by Big Data we live in.

Among the various methods that can be used for such analysis, I decided to focus on a form of AI Machine Learning technique named Deep Learning (DL) and to build a model based on it. The reason for this choice is not simply driven by the curiosity towards a model that has been recently hyped among the AI community, but by the fact that it might indeed one of the best tools to face such problems in the proper way. This because DL is built for specifically dealing with big amounts of data and perform complex tasks where automatic learning is a necessity.

After having explained how AI and more specifically DL models are built, I will use this tool for forecasting the market sentiment through the use of news headlines. The prediction is based on the movement of the Dow Jones Industrial Average (DJIA) by analyzing 25 daily news headlines available between 2008 and 2016. The result will be the signal that can be theoretically used for developing an algorithmic trading strategy. The analysis will be done on two specific situations (that I called A and B) that will be pursued throughout five time-steps: today (T0), today+1 (T1), today+2 (T2), today+3 (T3), today+4 (T4).

On the one hand, Case A will use a DL system to predict, for instance, based on the

news headlines published on T0, if the adjusted closing price of T3 is higher than the opening price on T3.

On the other hand, Case B uses a similar DL system to make an analogous prediction, but this time only the intervals will be considered, instead of the single days. Using the same example: It will be analyzed, based the news published in T0, whether the adjusted closing price of T3 is higher than the opening price of T0.

In order to test the models' applicability in real case scenarios, both case A and B will be implemented in two situations:

- Practical case 1: "Political election case" where we will explore how accurately the model would have performed right after the expression of the popular vote in a western country.
- Practical case 2: "Policies during Trump's presidency" where it will be investigated to see how well the algorithm would predict the DJIA movements based on some of Trump's most controversial policies during its presidency.

In the first chapter, we will start discovering what market sentiment is by examining the weaknesses of the Efficient Market Hypothesis and some concepts of Behavioral Finance. This will help us explain why and how it is reasonable to trade on news in a Big Data environment using Deep Learning techniques.

An overview of linguistics will be delineated in the second chapter with a focus on natural language processing that aims at analyzing many aspects of the written language. A particular attention will be given to the concepts of ambiguity and preprocessing, since they deal with all those procedures used for preparing the language that will be used as input for our model.

In the third chapter, a general introduction to machine learning will be presented, an idea that will be pushed to its limits in the fourth chapter when we will dive deeply into deep learning algorithms. The fifth chapter talks about the vectorization of words and the most popular forms of word embeddings nowadays.

Finally I will introduce the model itself, together with the analysis, the methods used and the results obtained in its practical implementation. In addition, a couple of sections will show the related work and the steps that could be done for improving the model's accuracy in the future.

Our journey concludes with the presentation of the doubts and suggestion on how deep learning techniques for systematic investment strategies could improve with further research, but also the general impact that this new technology might have in the financial markets.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Gaspari for the continuous support of my master thesis, for its patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better mentor for my Master studies.

Besides my advisor, I would like to thank and Prof. Rossi, Prof. Tamburini and Dr. Nicola Benatti for their insightful comments and suggestions on how to improve the model since pushed me to widen my research from various perspectives.

Contents

Chapter 1

Investor sentiment

In this chapter we will have an overview of the main strategies that are traditionally used for investing such as Fundamental and Technical analysis. Understanding them properly will help us explore a third way which is trading on sentiment. To do so we will analyze the flaws of the Efficient Market Hypothesis and introduce the basics of behavioral finance; this will allow us to on explain why it is possible to trade on alternative data, specifically news.

1.1 Investment approaches

When an investor approaches the stock market, he can refer to it by using four main points of view [3]:

1. Fundamental Analysis.
2. Technical Analysis.
3. Efficient Markets Hypothesis (EMH).
4. Behavioral Finance.

On the one hand, fundamental analysis is a method used to evaluate a stock by examining its most relevant economic factors. Technical analysis, looks on the other hand at various indicators (like trading volume) and uses complex mathematical formulas to try to predict market movements [3].

The EMH has been one of the most important finance concepts for years. Fama (1970) stated that financial markets are considered efficient when prices fully represent the totality of the available information. In Layman's terms, an average investor cannot consistently beat the market, which means that every effort in analyzing, picking, and trading securities to gain extra returns is pointless [32]. Behavioral Finance, at last, supports human fallability in the stock markets and therefore allows traders to make profit out of those biases.

1.2 The Efficient Markets Hypothesis's weaknesses

Despite its popularity, in the late-twentieth century the EMH began to lose support among academics when various mathematical models built upon it began to fail [3]. We have to be clear that these failures do not represent uncontroversial evidence that the EMH is wrong as a whole, but they point out that saying "prices reflect all information" is an incomplete assumption [3]. The simplest form of the EMH states that:

1. Investors are rational.
2. The trades of irrational investors are random and therefore cancel each other out.
3. Rational arbitrageurs eliminate the influence on prices of irrational investors even if they cluster together.

But the EMH goes one step further since it does not simply claim that information moves the financial markets, but that ONLY information does [6].

Arbitrage is defined as the concurrent sale and acquisition of the same (or similar) securities in different markets to make a profit [32]. If investors are rational, it means that they value each security by the net present value of its discounted future cash flow, which means that security prices should adjust at the same speed of the new information appearing to the scene [32].

Thus, irrational investors keep losing money until they disappear from the market. Moreover, this immediate reaction to new information, as well as the non-reaction to non-information (called stale information), plays a major role in the EMH's assumptions. There are three forms of EMH:

1. The **strong form** of the EMH says that information, regardless of public or private, is already incorporated in present prices, therefore there is no gain in trading based on those news [22].
2. The **semi-strong form** of the EMH says that since all publicly available information is incorporated in the current stock price, investors cannot beat the market unless they have some kind of information that is not readily available to the public [22].
3. The **weak form** says that stock prices reflect all the historical data and therefore, that technical analysis cannot be used to consistently profit from the market. If we cannot predict future prices on the basis of historical ones, then why should we look at them in the first place? nevertheless, fundamental analysis can still be used to profit from the stock markets [22].

Scholes said that the only way to obtain market efficiency is through arbitrage, which can only be done when perfect substitutes[1] are available [32]. Arbitrage becomes risky

[1]Perfect substitutes are goods that function just the same as the goods that have been compared to. For instance, Pepsi is the perfect substitute for Coca Cola.

when substitutes are imperfect; the phenomenon known as "arbitrage risk" (or "noise-traders[2] risk") comes from the possibility that the current mispricing becomes even worse instead of immediately disappearing, forcing the arbitrageur to a loss [32].

It is hard to prove that investors are completely rational; in fact, many investors react to useless information, follow passively the hints of money gurus, fail to diversify, implement the wrong models and have incorrect entry/exit strategies. Even more irrational are behaviors such as the "framing effect", which means that individuals get often influenced by how a situation or a problem is presented to them and can react differently [32].

The argument at the heart of behavioral finance is the following: investors are not rational and therefore their judgment is influenced by biases, prejudices and fears [3]. In fact: "The volume of statistical evidence of the phenomena of under and overreaction in security returns is enormous." [32]. Another line of defense of the efficient markets theory is that irrational investors trade randomly, and therefore those irrational behaviors will cancel with one another.

Unfortunately, there is evidence that noise-traders are not fully irrational and those who can be inserted in this category do not perfectly cancel each others out anyway [32]. This is due to the fact that investors often move in herds, since noise-traders seem to follow each others' mistakes [32]. As Charles Kindleberger[3] once said: "There's nothing so disturbing to one's judgment as to see a friend getting rich" [6].

Humans, including professionals such as portfolio managers, tend to make bad predictions under certain conditions [1]; this is mainly due to the fact they have and pursue their own interests while managing other people's finances, such as avoiding excessive losses not to lose their jobs [32].

"Shiller (1981) showed that stock market prices are far more volatile than could be justified by a simple model in which these prices are equal to the expected net present value of future dividends" [32]. This statement collides with the assumption that it is impossible to make excess gains using historical price information. Moreover, prices can indeed be influenced by causes that lack a clear explanation and can appear unknown. The most evident example of this is represented by the crash of 1987 when the Dow Jones Industrial Average (DJIA) fell by 23% without any clear news related [32].

If we have all this evidence...why have researchers failed to report it? and why haven't they challenged the EMH right away?

After all, any theory can get high grades if it can take credit for its successes , while

[2]The so-called "noise trader" is a term used to describe investors who trade without using any kind of market analysis, nor following professional advice. They tend to be impulsive, overreact and follow trends.

[3]famous economic historian.

its failures can be labeled as meaningless "exceptions" that are allowed to be ignored. It is like saying: "Despite a few exceptions, the Russian Roulette is a safe and fun game for all the family to play" [6].

On the one hand, a cause could have been the almost religious faith of EMH by its followers, which made it difficult for opposers to be taken seriously until the 80s [32]. On the other hand, Summers (1986) argued that lots of experimentation linked to the EMH cannot discriminate against possible forms of market inefficiency (which is, scientifically, a more satisfactory explanation). The rise of behavioral finance slowly restricted the applicable role of perfect market efficiency to a few limited cases [32].

1.3 Behavioral Finance

The idea of market sentiment was firstly introduced by Keynes with his "beauty contest analogy". He argued that investors are involved in selecting the most "beautiful" securities based on how popular they were, rather than independently analyzing the stock's value. This idea of the "beauty contest" explains why some stocks are incomprehensibly overvalued and others unexpectedly undervalued [3].

At the most general level, behavioral finance is the study of human fallibility in the financial field. Behavioral finance theory (BFT) has two major foundations:

1. **Limited arbitrage**: As we have already said, arbitrage is far from perfect due to many imperfections such as lack of perfect substitutes, arbitrage risk, reactions to non-information and slow adjustments; this helps us explain why prices do not necessarily react to information by the right amount.

2. **Investor sentiment**: the theory of how real-world investors actually form their beliefs and valuations, and more generally their demands for securities [32].

The core concept so far is that the opinions of noise traders are unpredictable; thus arbitrage requires bearing the risk that the security will keep moving in the wrong direction even more, forcing the arbitrageur to lose money [32]. As John Maynard Keynes said: "The market can stay irrational longer than you can stay solvent" [6]. In fact, stocks that had a long track record of good news tend to become overpriced, a phenomenon called overreaction; and have low average returns afterwards since prices, on average, return to the mean [32].

Another distortion is the fact that people see patterns where there are none [32]. For example, it has been proven that when investors are surprised with good news multiple times, they tend to automatically revaluing a company instead of considering the likelihood that those few positive surprises are the results of chance [32].

This model is motivated by two phenomena documented by psychologists [32]:

1. **Conservatism**: Individuals are slow to change their beliefs when new evidence come out. *"It turns out that opinion change is very orderly; a conventional first approximation to the data would say that it takes anywhere from two to five observations to do one observation's worth of work in inducing a subject to change his opinions"* [32].

2. **Momentum** happens when the so-called "positive-feedback investors" buy after a price increase and sell after the stock market falls [32]. This phenomenon can be enhanced by stop-loss orders or simply by investors which would be unable to meet margin calls and are therefore forced to liquidate their positions. This is the reason why the so-called "price bubbles" start building up [32]. Investors such as George Soros have been very successful by betting on future crowd behavior: He would cause an initial price rise in certain stocks, this increasing price trend would attract the appetites of uninformed investors that drove prices even higher. At some point these price rallies must stop and stock prices would fall [32].

There is also evidence that if today's prices fall the market will become more volatile in the near future, while rising prices cause the opposite effect and calm the stock markets [6].

All of these market inefficiencies were often treated as "anomalies" by EMH supporters since they usually disappear; nevertheless they should be looked at as **regularities** since the keep repeating over and over again. Another interpretation is that arbitrage can catch up with some of them (with the most evident at least), however it can take a long time to do so, especially when the disturbing sentiment is powerful [32].

We now arrive to an important question, do these inefficiencies have real consequences on a firm? or we are simply talking about a mere redistribution of wealth from impulsive to screwed investors? Let's say we live in the world of the EMH, in this case Modigliani-Miller's Theorem[4] states that the value of a business is independent from the firm's capital structure. However, when the markets are inefficient and investors have these biases it means that firms can take advantage of these factors too.

Suddenly selling equities in the primary market looks like selling books door to door [32]. Why? because from the firm's perspective it pays good money to alter its financial structure, even slightly, to take advantage of the investors' sentiment and to make sure that its securities are more appealing to the investors' desires [32]. This line of reasoning applies, for instance, to IPOs, dividend strategies, buyback policies and so on.

After having understood all of these biases, that appear to be more and more as "regularities" in the market, maybe we can learn something new by accepting them.

[4]the main proposition of corporate finance

1.4 The Adaptive Market Hypothesis

Academics tried to fill the holes of the EMH by adding some aspects of behavioral finance to it. The case that we will be considering is the **Adaptive Market Hypothesis (AMH)**. While the EMH specifically states that predicting price movements is not possible due to their randomness, the AMH allows for forecasts in the short-run [3].

"By providing for the predictability of markets, the ability for market participants to learn and adapt to market changes, competition between market participants, and the existence of profit opportunities within the market, the AMH is an excellent theoretical foundation to use as a framework to study predictive approaches to the marketplace" [3].

The AMH explains the investors' irrationality by noticing that they may enter the market for different purposes; and that those reasons may change overtime. Moreover, the AMH describes some important implications [3]:

1. Existence of profit opportunities.
2. Competition will ultimately get rid of those arbitrage opportunities.
3. Simple investment strategies will erode faster than complex ones.
4. Survival of the fittest, where innovation plays a major role.

1.5 Trading on news

Financial markets in the digital age are more connected than ever before, in fact billions of users can come up with new investment ideas using the internet. These modern forms of financial hubs can be seen as social networks where individuals continuously share tons of digital content [29].

In 2012 Gartner[5] defined Big Data like this: "Big Data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision-making, and process automation".

In Big Data environments we have the necessity to analyze and act quickly and many times. Unfortunately, to exploit Big Data (especially the unstructured ones like text information) we have to face issues such as trustworthiness, high dimensionality, form, variety and efficiency [29]. Thus, one of the most impelling issues is how to deal with these constraints, especially when it comes to decide when, how and with what security we should enter a trade [29].

[5]https://www.gartner.com/it-glossary/big-data/

Big Data showed disruptive potential and there is a high incentive in looking for technologies that can crunch those huge amount of information and extract insights. The most common methods are Rule-Based Systems (RBS) and Artificial Intelligence (AI) [25], but given the high effort involved in implementing the first one by trial-and-error there is a strong preference for algorithms that can learn by themselves.

Definition 1.5.1. Sentiment analysis (or opinion mining) is a form of data mining technique that measures the people's opinions through computational linguistics, which is used to process and analyze data. These information quantify the general public's opinions towards products, people or ideas [34].

Opinion mining is a research area that has been revived in the last decade. This hype has been boosted due to the rise of social media and of the exponential increase in cheap computing power availability and by the simplicity in finding databases in a Big Data environment [3].

Semantic technologies, once added to the equation, pushed an increase in accuracy in the decision-making process in many use cases [25]. The financial world is not exempt from this and is becoming the more and more a knowledge intensive domain [19]. In fact, computational linguistics in finance has gone a long way in revolutionizing the way we manage information, by helping financial players to make better decisions [19].

Thus, opinion mining is an especially popular issue, especially for modern trading firms [19]. Market sentiment is the general hope of investors to anticipate price trends as "Bullish[6]" or "Bearish[7]" to profit from them. This sentiment is not easy to define or express, but it is usually seen as the combination of many elements such as politics, seasonal factors, financial reports, and so on [29].

The analysis of financial news through algorithms is a particularly hot topic: analysts' predictions are often based on what they read in the news [19], is it possible that there is predictive power in the stock market's behaviour based them? Yes, this fact has been especially proven in the short term [13]. In most of these cases, to determine the polarity of a text, it is assumed that a negative message in a news announcement correlates with negative terms and the other way around [19]. For example:

1. Positive sentiment: 'profits'.
2. Negative sentiment: 'depreciation'.

One of the biggest difference between market sentiment problems and linguistics ones is that the ladder has some guarantee of having a learnable structure[8]. On the contrary,

[6]An investor is considered Bullish if he thinks that in the future the stock price will increase and therefore takes a long position on the security.

[7]Oppositely, a Bearish investor expects a price decrease over time and will short the stock.

[8]In the form of grammar rules

markets do not necessarily show such characteristic in such an obvious manner [28]. Thus, the idea of analyzing the market in the same way we encode the semantics of a paragraph seems plausible [28].

Assuming the structure is there, we reject the common assumption that "positive financial sentiment" = positive words and vice-versa. The reason is that we do not know if that is really the case, not in 100% of the situations and we want the model learn autonomously and without constrains.

The recent popularity for AI, in particular of Machine Learning (ML) techniques is given by the fact that they provides systems that have the capacity to automatically access data and improve without human intervention. In particular, the purpose of this work is to implement a particular form of ML, named Deep Learning (DL), that makes wide use of the so-called "hierarchical learning" an approach which takes direct inspiration by how human brain works [29].

Deep Learning is the computational technique that has recently achieved remarkable results in various fields such as speech recognition, computer vision and text analysis. Why? because when compared with more simple Machine Learning algorithms as shown in Figure 1.1, it shows keeps improving thanks to its incredible capacity to crunch Big Data. In fact, while traditional models, at some point, reach a plateau DL give increasingly better results the more data they can train on.

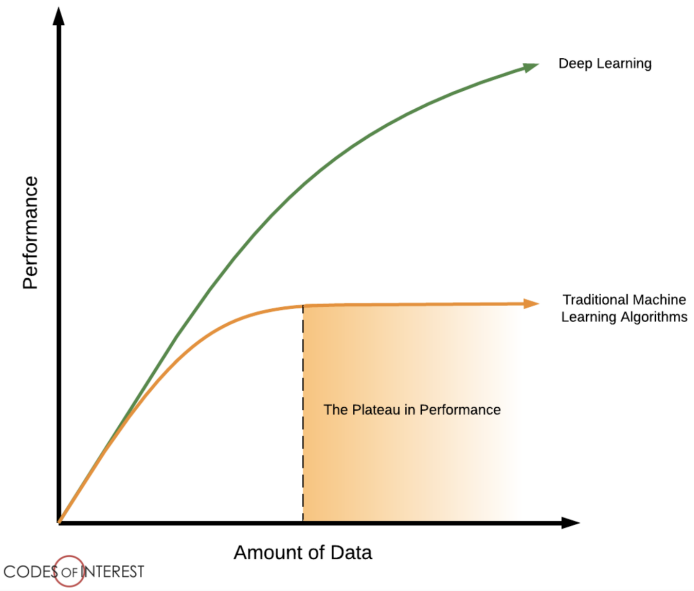

Figure 1.1: Deep learning performance compared to other models [2]

A DL algorithm will use this same intuition to extract a so-called "market signal", whose function is to make us understand how the stock market will react to specific

news. To find such signal, DL extracts non-linear patterns that will ultimately be used to make a prediction on how the market is likely to move [29].

Deep Learning is considered a black-box since most part of its work is done in the background while the user needs to decide only a few factors. Thus, it quickly gained popularity as an easy-to-use tool that can accomplish complicated tasks. DL can be used to extract nonlinear features in Big Data environments that are then used for practical implementations [29]. As the spread of analytics increases day after day in our society, DL become more and more relevant to face Big Data related problems.

From this discussion we understood that the AMH will be the theoretical basis for our research and that, in the next chapters, we will dive into this DL to try to consistently reduce the biases highlighted in Section 1.3 [29]. Before jumping into the actual mechanisms of DL it is necessary to have an overview of linguistics, of its components and on how it has to be dealt with before feeding it as input to our AI algorithm. [29].

Chapter 2

Natural Language Processing

In this chapter we will focus the different aspects of language in order to fully grasp its complexity. This will help us build an approach that we could effectively apply later on from a computational perspective.

2.1 What is natural language processing?

Whatever we say, read, write, or listen to is mostly in the form of natural language (NL), this is because it is our way to express thoughts and feelings. We are great at producing, understanding and interpreting languages despite their complexity. In fact, the human's way of expressing concepts is not just articulated, but also continuously evolving [23].

Natural language processing (NLP) is a broad field of artificial intelligence which is focused on finding interactions between computers and the human language. In fact, NLP is a mix of three different studies: linguistics, artificial intelligence, and computer science [23]. **Natural language understanding** (NLU), in particular, is a subcategory of NLP which represents the process of making a language tool understand what humans communicate.

In our case, the aim of sentiment analysis is to mine the public's opinion and create a profitable quantitative trading strategy based on that signal. To do so, we need to work with huge amount of written text and on a machine that can understand them.

2.2 Basic linguistics concepts

In this section we will clarify how words and sentences are structured by exploring the underlying mechanisms of linguistics.

Semantic analysis

Semantic analysis is the process of linking syntactic structures, such as documents, paragraphs and sentences at the document level. Thus, the task of semantic analysis is getting the proper meaning of the sentence [35].

Syntactic analysis

Take the sentence: "president best the is Trump US ever", this sentence does not have logical meaning, and its grammatical structure is not correct. Syntactic analysis, through grammar rules, studies the correct logical meaning of sentences [35].

Lexical analysis

Lexical analysis approach the NL analysis from a slightly smaller perspective since it is the process of breaking down a text into words. The most important concepts in this field for us are:

- **Tokens** are created by breaking a sentences into small pieces. The process of deriving a token, which consists of identifying the boundary of sentences/words, is called **tokenization**.
- A **part of speech (POS)** is a category of words or lexical items that have similar grammatical properties and have similar behavior; they are usually tagged to each word. POS categories are verbs, articles, nouns, adjectives, etc.
- **Lemmatization** is the process of grouping together words in a dictionary-like form so that they can be analyzed as a single item called lemma [35]. For example the lemma of "unhappily" is "unhappy".
- **Corpus**[1] is a collection of written or spoken natural language material, stored on computers, and used to find out how language is used in a specific context [35]. Corpus analysis, in fact, consists in analyzing, manipulating, and generalizing the corpus through statistical techniques to have a better understanding of the whole dataset [35]. In any NLP application, we need data and a corpus provides precisely the necessary quantitative information that are used to build NLP applications [35].

One of the biggest challenges of NLP deals with deciding the right type of data: in particular concerning the **adequacy** and **availability** of **high-quality** data that we need to work with.The meaning of individual words can, in fact, be quite slippery. Think

[1]Plural "Corpora". In some cases, a corpus is referred as a "dataset".

for example at hyponyms[2], hypernyms[3], homonymy[4] or polysemy[5].

Morphological analysis

Morphology is the study of the smallest elements that form words and specifically deals with their grammatical analysis of how words are formed using morphemes. A word is the single and tiniest element of a sentence that carries meaning. Its structure is shown in Figure 2.1:

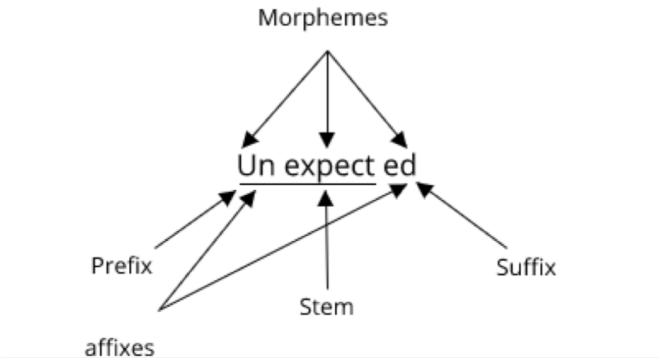

Figure 2.1: Morphological structure of a word [35]

Thus, a **morpheme** is the smallest meaningful morphological unit available. As shown in the Figure 2.1, morphemes in general are affixes and they can appear as prefixes[6], suffixes[7], infixes[8] and circumfixes[9].

The stem is the most important concept to grasp here and will appear later in this section; it is defined as that part of a word that an affix is attached to. For example the stem of *unhappily* is *happy*.

2.2.1 Ambiguity

NL can be slippery field for humans, take Figure 2.2 for instance, it is straightforward that this intrinsic ambiguity is what makes NLP even more problematic for computers to master [23]. There are not only hundreds of different dialects and languages, but also unique collections of grammar and syntax rules among them that change over time.

[2]A word of more specific meaning than a general term applicable to it. For example: yellow, red, green.

[3]A word with a broad meaning constituting a category into which words with more specific meanings fall. For example: color.

[4]Each of two or more words having the same spelling or pronunciation but different meanings and origins.

[5]The coexistence of many possible meanings for a word or phrase.

[6]A word/letter/number placed before another.

[7]A morpheme added at the end of a word.

[8]A morpheme that appears inside a stem.

[9]A circumfix is an affix made up of two separate parts which surround and attach to a root/stem.

Figure 2.2: Ambiguity [26]

For example tokenization can be both at the sentence or word level, depending on the task assigned and they are performed together, we might encounter some troublesome situations [23]:

- Take a sentence "Harry Potter was written by J.K. Rowling. It is an entertaining one." the machine is tempted to tokenize the phrase after "K." , given the space after that point, but that's not what we want!
- Consider a phrase with a mistake inside: "Chess is a difficult game.I have never won once." in this case, we want instead to split the two sentences correctly, regardless of the absence of a space after "game.".
- If you have a sentence such as "The white house is huge" it can be ambiguous, since the statement can be seen both in the context of the presidential "White House" in the USA, or a house nearby, whose color is white [35].

Long story short, tokenizing words, such as "don't", can be pretty painful to deal with; however, it is possible to improve the accuracy of our tokenization process by writing exceptions individually [35].

Word sense disambiguation (WSD) is therefore one of the major battles for NLP since ambiguity can appear at a morphological, lexical, syntactic, or semantic level in all of their nuances [35].

2.3 Preprocessing

Only around 21% of the data available on the web appears us in a completely structured form [23], the rest is raw written text. Data preprocessing describes any type of action performed on raw data with the aim of transforming them into a format easier to access and understand.

Text preprocessing involves three main steps: **Noise removal**, **text normalization** and **object standardization** with the common purpose of simplifying the model.

The first one, consists in cleaning those parts of text which are irrelevant to the content such as URLs, acronyms and slang [23]; usually by preparing a dictionary of

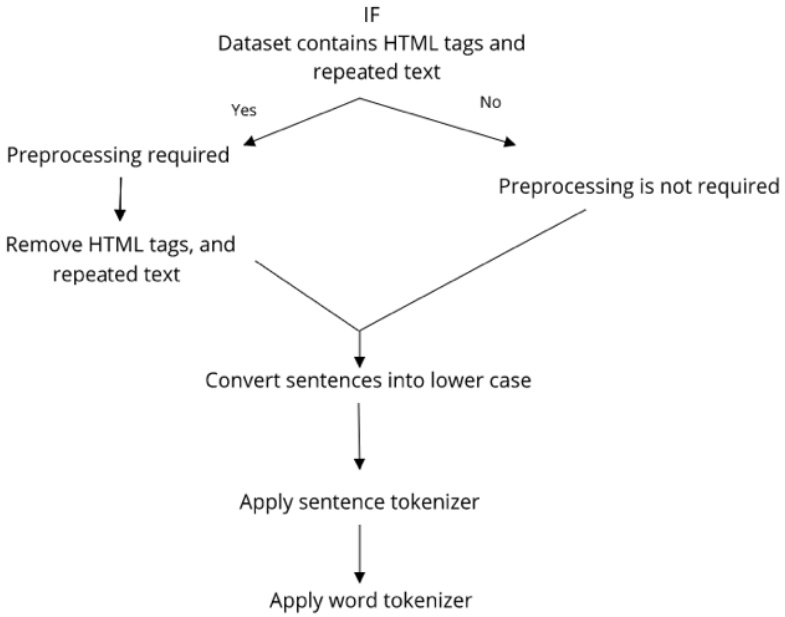

Figure 2.3: Is Preprocessing required? [35]

undesired elements to eliminate all at once.

The second one consists in trying to reduce the overall dimensionality of the corpus by transforming the tokenized sentences arbitrarily. There are various techniques used for these purposed that can be more or less invasive, take lemmatization or stemming for that we have previously encountered for example [23].

2.3.1 Common preprocessing procedures

Given a raw text file we need to define its raw data as variables. If the amount of available data is small (those from a tweet for example, that can contain a maximum of 280 characters), then we could, for example, assign that specific data to a local string variable directly.

In this section, except for the tokenization procedure that we have previously encountered, we will see the most common tools that are used in the literature to manipulate text datasets at our advantage, namely [35]:

1. Stemming or Lemmatization.
2. Stop-Words and Rare-Word Removal.
3. Case Lowering.

Lemmatization

Lemmatization takes those inflected word forms and returns them to their base form: lemma. To achieve this, you need some context of the word to use, like whether it is an adjective or a noun.

Stemming

Stemming, on the other hand is a more simplified way of obtaining the same result and does not take the context into account [23].

From a computational point of view they serve the same purpose; however stemming has lower accuracy, but a higher computational speed than lemmatization. These two techniques are a strong form of text manipulation that should be used with caution [35].

Remark. What does it mean that stemming has a lower accuracy? Let's use a few examples. In most case stemming and lemmatization give the same output, for example they transform "studying" into "study". However, there are cases where the outputs differ, like in the word "studied" a lemmatizer will output again the word "study" correctly as we expect, while a stemmer will output "studi".

Stop-words

Stop-Words are words that get filtered out. The reason is that they have very little meaning, such as "and", "the", "a", "an", and similar words. **Stop words removal** is important since it helps getting rid of stop words and therefore reduce unnecessary elements.

A list of stop words that can be found in the NLTK API[10], for the English language, are namely:

['I', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't",

[10]The most widely adopted Python Package in use today for NLP.

'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

On the opposite side we have the procedure called Rare-Words removal which consists in allowing our model to accept only a limited amount of tokens by eliminating the most infrequent ones.

Case lowering

Since these kind of NLP applications are case sensitive, converting all the data to lowercase is a simple standard procedure that drastically reduces the computational effort of our database [35].

2.3.2 Selective preprocessing tools

Regular Expressions

Regular Expressions (RegEx) are a powerful tool when the database is extremely noisy and we want to manipulate specific information from any text. It helps us find or replace specific patterns from a sequence of characters. [35]

For example, if I want to eliminate all the people tagged from a tweet I know that they are formed by a @ symbol and then some letters until the next space. Therefore we can tell the regular expression to find and eliminate all those words that start with that specific symbol, until the closest empty space. Other use cases can be the removal of numbers from a text, or of all the punctuation from the corpus.

Lookaheads

When we start preprocessing, sometimes you need to do some ad-hoc customization:

Positive lookahead

Consider a sentence: "I play on the playground". You want to extract "play" as a pattern but only if it follows "ground". In this situation, you can use positive lookahead.

Positive lookbehind

Consider the sentence we used before, but now you want to extract "ground", if it is preceded by the string "play".

Negative lookahead

Consider the same sentence; you want to extract "play" only if it is **not** followed by the string ground.

Negative lookbehind

Consider the same sentence for the last time. Now you want to extract ground only if it is **not** preceded by the string play [35].

To sum up, when it comes to preprocessing, despite the multitude of techniques that can be applied; you need to have a deep understanding of the text that you are working on and what you are trying to do with it. To do so, you need to be able to ask the right questions.

2.4 Feature Engineering

To analyze preprocessed text data using AI, you must convert them into features. What is a feature exactly?

Let's make an example to simplify our journey: suppose that we want to create an algorithm that can predict the house prices in New York. Features that can be used are for example its dimension, the number of bathrooms or whether it has a garage.

Features in a AI model are individual measurable properties used as input to obtain an output [35]; therefore any attribute can be a feature as long as it is measurable and useful! These features are constructed using several different statistical methods that we are going to reveal soon [23]. In the NLP case, for example, the features that will be taken into account are the words, which can be measured by the number of times that they appear in the corpus.

Definition 2.4.1. Feature engineering in NLP problems is the process of understanding and choosing from a corpus features that are most useful for your problem [35].

Feature engineering is crucial to solve NLP problems, since if you are able to derive good word features, then you will have an easier life in terms of:

1. Flexibility.
2. Accuracy.
3. and you will obtain good results even in sub-optimal algorithms.

At the same time, when you are about to apply feature engineering tools, you need to be aware of the fact of its drawbacks and challenges [35]:

- Complexity in the choice of good features.
- Time-consuming process.

- Usually, choosing the right features requires expertise in the domain that you are about to work on.

Vectorizing text will be one of the most challenging task, for this reason it will analyzed more in detail in Chapter 5. [35].

Chapter 3

Machine learning

In this chapter we will start building from the ground up the various forms of Artificial Intelligence (AI) to have a taste of their strengths and weaknesses. This will help us to have a better intuition on why we chose to use deep learning for our purposes in the next chapter.

3.1 The basics of Artificial Intelligence

When we talk about AI in this context we refer to an area of computer science that wants to empower computers with human-level intelligence [35]. What do we exactly mean with machine intelligence?

It is a form of AI characterized by being automated, which is the case when a machine acts without human intervention [35] and expresses some form of intelligence; which is a very broad term, since it requires skills such as abstract reasoning, learning, problem-solving and much more [35].

Therefore, when we talk about machine intelligence, humans expect computers not only to automate tasks, but ultimately to understand our emotions and to perform the majority of our tasks [35]. AI when needs to be defined, as shown in the Figure 3.1, can be regarded as a hierarchy involving:

Machine learning

Machine learning (ML) is the broadest term of all; it is related to intelligent systems that learn specific patterns and improve with experience, in particular from real-time or historical data [35].

Machine intelligence

Machine intelligence goes one step further, since these kind of algorithms learn patterns from data with extremely low human intervention [35]. This is the current stage of

development of our technology and artificial neural networks (ANNs) techniques, such as deep learning (DL), are the main protagonists of it.

Machine consciousness

Machine consciousness is the level we do not have reached yet and it is the future of AI, where machines would also learn from their own experiences, do abstractions [35].

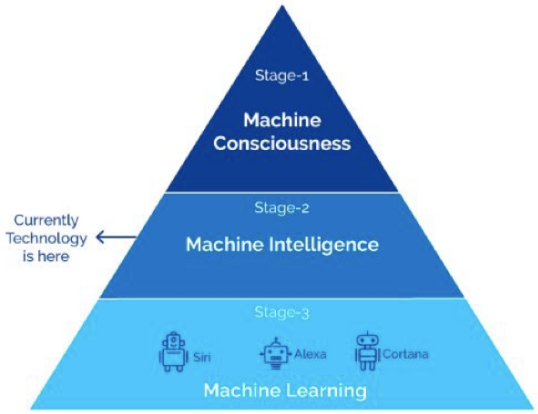

Figure 3.1: Stages of AI method [35]

3.1.1 Natural Language Understanding in ML

Figure 3.2: ML approaches [35]

Traditionally, programming has been all about defining all the steps to reach a certain outcome. ML flips this approach upside down: the outcome is "shown" first and the program learns the steps to achieve it.

Let's make an example with images, since they are much more intuitive: Suppose that you want to write a program that will help you classify cats images from non-cat figures. You start to code that a cat has two eyes, the fur and so on until the very last minimal detail. Sounds quite time-consuming and prone to error doesn't it? Thanks to machine learning, instead of writing many lines of detailed code, we load the machine with cat images and obtain a better result [35]. By looking at Figure 3.2 we can have a visualization of the most popular AI approaches.

Going back to the text case, natural language understanding (NLU) consists in applying various ML techniques on NL in order to understand the content of a text document [35]. Before moving on with some forms of ML, let's clarify some terminology [35]:

1. Instance: input.
2. Concept or hypothesis function: a function that maps input to output.
3. Target concept: the idea that we are trying to find.
4. Hypothesis class: Is the class of all possible functions that can help us to identify the function that we need.
5. Training dataset: A list of examples (labelled input) that we provide to the ML model so that it can improve itself.
6. Candidate: The predicted target concept that the ML-model outputs, which can be correct or wrong.
7. Testing dataset (or Validation set): A database that the ML-model has never seen before. If we use data that the machine has already seen in the training set, as part of the testing set, this last one becomes completely useless [35].
8. Loss (or error) function is used to measure the inconsistency between the value predicted by the ML algorithm and the real one. The lower the value of loss function is, the more robust is the model.

3.2 Machine Learning Categories

ML can be divided in three[1] main categories as shown in Figure 3.3:

1. Supervised Learning.
2. Unsupervised Learning.
3. Reinforcement Learning.

[1]Consider that reality is always more complex than it looks. In fact, many of those categories may overlap, merge or have a few common traits.

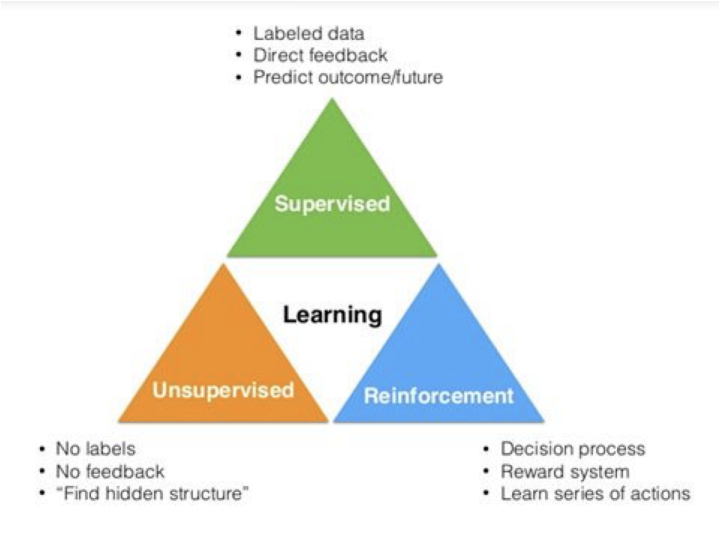

Figure 3.3: ML classification [35]

3.2.1 Supervised learning

The essence of ML models is the creation of mechanisms that can look at examples and produce generalizations [15]. The supervised learning (SL) case needs a labeled database to work, from which the ML algorithm sees the relationship between data and its labels.

An example widely used in literature is the classification of spam and not-spam e-mails. SL uses the labeled examples that receives and generates a model whose goal is to predict to which category each new upcoming e-mail will belong [35].

3.2.2 Unsupervised learning

Unsupervised learning takes unstructured data as input, which means that they not receive any form of human intervention. Therefore, the machine needs to independently learn the necessary data structure (which is usually raw, noisy, and complex) to tackle its task (Figure 3.4) [35].

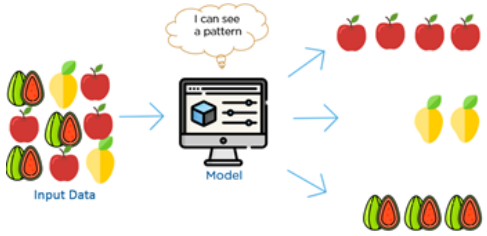

Figure 3.4: Unsupervised learning[2]

Note that the biggest drawback of unsupervised learning is that in some cases it can be extremely hard to understand what the output of the algorithm actually means. Therefore, sometimes it is simply less time-consuming to manually label a dataset rather than trying to figure out what the code is doing.

3.2.3 Reinforcement learning

This type of learning is mostly used in the area of robotics or to develop bots to play games for example. Reinforcement Learning (RL) is a form of dynamic programming that trains algorithms using a reward-punishment system with minimal human intervention. A RL algorithm (called agent) learns by interacting with its surrounding environment: it gets rewarded if it performs a task correctly and gets penalized if it does not [35].

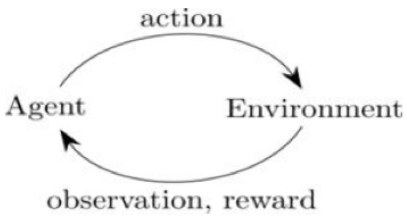

Figure 3.5: Reinforcement Learning [35]

3.3 Basic Machine Learning concepts

3.3.1 Error Function

An Error (or cost or loss) function gives us an idea of how good a ML model performs. The cost function is used to inform us how far away the predicted data are to the true values and the closer they are, the better the forecast of our algorithm will be in real implementations [35].

There is no one-fits-all error function it depends on the problem that we are trying to solve which can be either a **regression**[3] or **classification**[4] task [35]. The ladder is the one we care about, since our model is based on a binary classification problem we need a Binary Cross-entropy loss function.

[2]https://www.quora.com/What-is-the-difference-between-supervised-and-unsupervised-learning-algorithms

[3]Regression is a statistical process that tries to identify the relationship among different variables. First, the dependent and independent variables (predictors or features) need to be identified [35].

[4]Classification is used to find a category for each observation and is the approach that we will use for our purposes. In fact, we want our model to decide if a certain text has either a positive or negative sentiment. The algorithm that implements all of these operations is called a classifier [35].

Cross-Entropy

The cross-entropy loss (or log-loss) function is used in classification problems and measures the performance of a model whose output is a probability that goes from zero to one [10]. Cross-entropy, in binary classification problems, is called Binary Cross-entropy and is known as:

$$-(y \log(p) + (1 - y) \log(1 - p))$$

3.3.2 Accuracy Metrics

The accuracy metric is determined when the learning phase is complete and has the same importance of the error function. At this point, the validation samples are fed to the model and the number of mistakes the model makes are recorded and, after a comparison to the true targets, a percentage of correct classification is computed.

In Layman's terms: if we have a model that needs to classify cats and dogs with 100 test samples, and it classifies correctly 75 of them (predicts "cat" when it sees a cat image and predicts "dog" when it sees a dog image) then it has a 75% accuracy. Why do we need both the error function and accuracy to evaluate a ML model's performance?

The reason is simple, they are both useful for two different reasons: while the error function is particularly useful in the training phase[5], accuracy is more intuitive and shows in real case scenarios how the algorithm is performing.

3.3.3 Overfitting

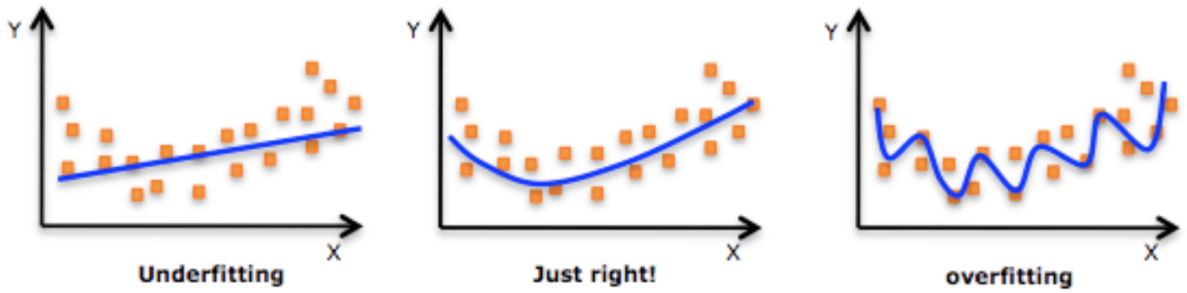

Figure 3.6: Overfitting and Underfitting [6]

Overfitting means high **variance**. It appears when you have a high accuracy in the training set but a low one in the testing set. A machine that overfits tends to act like a student that has been memorizing the preparatory tests to such extent that he is no longer able to generalize the theoretical concepts he needs to score high at the real final

[5] since it shows you whether your model is improving or not over time
[6] https://www.datarobot.com/wiki/underfitting/

exam [35].

More formally, suppose we have a dataset and we plot all the data points on a two-dimensional plane. Now we are trying to classify the data and our ML algorithm draws a decision boundary in order to classify the data.

At the center of Figure 3.6 shows a model with the right fit among the datapoint. However, when you look at the right-hand side of the same Figure 3.6 you notice that the ML-model does really bad. This is what happens when an algorithm overfits, it has high variance and is not able to generalize unseen data [35].

3.3.4 Underfitting

An algorithm underfits when, regardless of how much you train your model, you do not see much improvement this means that the bias is too high [35], as shown in the left side of Figure 3.6.

Remark. Remember that the total error of a model is made by:
Total error = Bias + Variance + Irreducible error
As you might have guessed from its name, we cannot get rid of the latest, therefore you should concentrate on minimizing the first two by finding the right balance.

Machine learning is the story of the eternal battle between overfitting and underfitting and it is the coder's ability to find the right equilibrium between them that separates a good model from a great one.

3.3.5 Curse of dimensionality

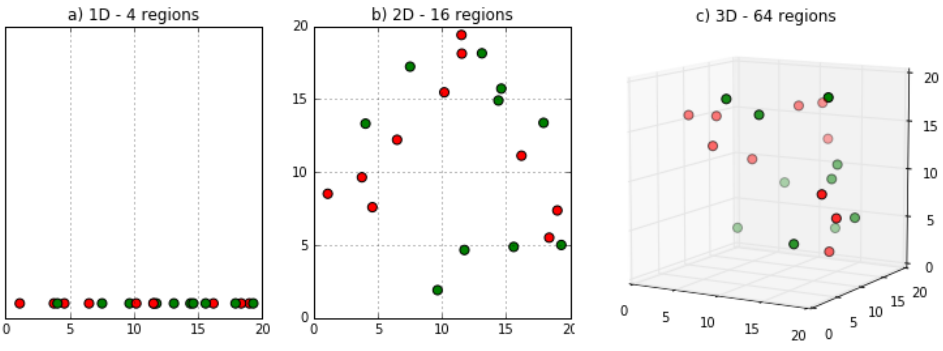

Figure 3.7: Curse of dimensionality[7]

Suppose you have a line, 1-dimensional feature space, and if we put ten on it, each point takes one tenth of the line space. Let's suppose that you now have a 2-dimensional space, in this case we need at least 100 points to fill the 2 dimensions. In the case in

[7]https://www.kdnuggets.com/2017/04/must-know-curse-dimensionality.html

which we have a 3-dimensional space, we need 1000 points; and so on for higher dimensional spaces [35]. The curse of dimensionality says that, as the number of features (or dimensions) grows, the amount of data that we need to obtain an accurate generalization explodes accordingly [35].

Dimensionality reduction is a very useful concept in Machine learning used to remove unnecessary features [35]. In the NLP case, the simplest way to reduce dimensionality is by implementing procedures aimed to reduce the number of words fed into the model. The conclusion is that the number of features that need to be fed into the ML-model should be kept at their bare minimum. As Einstein once said:

"Everything should be made as simple as possible, but not simpler."

Chapter 4

Deep Learning

Deep learning (DL) is an advanced ML method that uses Artificial Neural Networks (ANNs) with many layers, for this reason it is addressed as deep neural networks (DNN). With DL, the line which separates supervised and unsupervised learning fades away due to the fact that these algorithms perform really well with both labeled, unlabeled and mixed databases [35].

4.1 Basics of Neural Networks

The idea of ANN takes inspiration by the way the human brain works: in Layman's terms, we want them to perform three main tasks [35]:

1. Receiving the signal.
2. Deciding to pass the signal to the cell body.
3. Sending the signal.

To sum up, we want to build a model that can deal with a lot of data and, depending on the computational power that we have, see it solving many complex problems much faster than humans would ever be able to do. Seems quite a challenging task, let's take a step back first... To be honest, the concept of NNs is older than you might expect [35].

4.1.1 The first computational model mimicking the neuron

The birth of this concept came in 1943, with the McCulloch-Pitts model that introduced the first computation model of such type, that was indeed a simple neuron. The model was meant to receive binary inputs, multiply them with their relative weights and sum them together. The following neuron structure takes as input some values (x_1 ... x_k) and their relative weights (w_1 ... w_k) that are summed together [35]:

$$u = \sum_{i=0}^{K} w_i x_i$$

However, the introduction of the concept of "activation function" to really made the difference [35]. The activation function, if compared with the processes in the human brain, plays the role of generating the spike that allows the information to pass.

$$y = f(u)$$

Nowadays there are tons of activation functions[1], but this primordial version of the neuron had a simple step-function: as shown in Figure 4.1, if the sum exceeds a certain threshold, then output is one otherwise is zero.

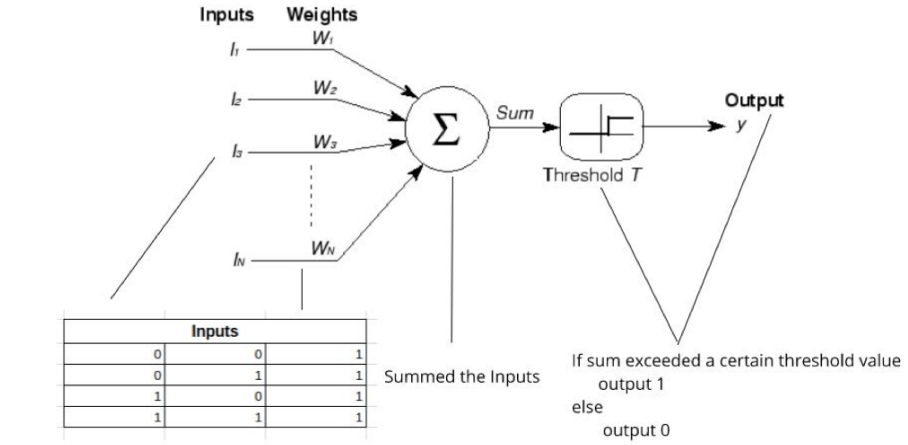

Figure 4.1: First neuron model [35]

Why is the activation function important? Because it gives the neuron, and by consequence the overall NN, the so-called Non-linearity property which allows them to become universal function approximators[2].

Non-linearity is a relationship where a certain variation of the inputs does not lead to a proportional change of the output. Non-linearity is a common problem when analyzing cause-effect relations because, when left uncontrolled, could lead to unexpected outcomes[3].

[1] We will see some of them in a specific section.
[2] https://www.investopedia.com/terms/n/nonlinearity.asp
[3] https://www.investopedia.com/terms/n/nonlinearity.asp

4.1.2 Basic weights initialization

We have explained that such a model multiplies the input with the relative weights; but when the model starts for the first time, how are the weights initialized? The answer is **randomly**, but we can choose the distribution that these random weights should follow when initialized. In most cases, the initial weights are:

1. Zeros.
2. Ones.
3. Constant value decided by the user.
4. Randomly generated using a Normal distribution.
5. Randomly generated using a Uniform distribution.

4.1.3 Perceptron

Later on, in 1957 Frank Rosenblatt criticized the McCulloch-Pitts model and introduced the concept of perceptron, also called a single-layer neural network [35].

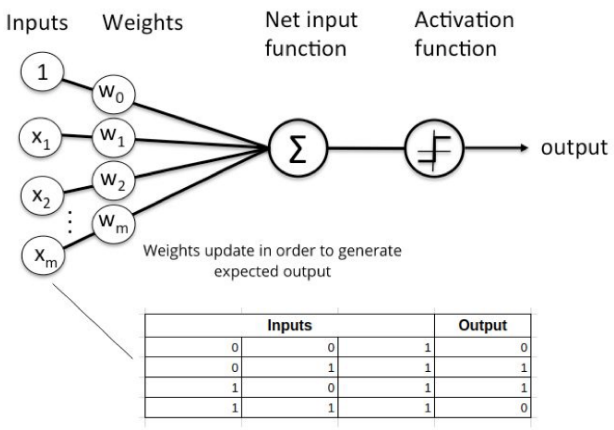

Figure 4.2: Perceptron [35]

Given a training set, the model should learn a function by a continuous increase and decrease of the weights in order to improve the accuracy at every iteration through the so-called training process.

Searching over the set of all the existing functions is a difficult task, therefore we need to restrict our scope of action by using smaller sets[4], but by doing so, we added an

[4]that we name hypothesis classes

"inductive bias" to our model that must be taken into account. The most commonly[5] used hypothesis class has form [15]:

$$NN_{\text{Perceptron}(x)} = f(x) = xW + b$$

$$x \in R^{d_{in}}, W \in R^{d_{in} * d_{out}}, b \in R^{d_{out}}$$

which is a high-dimensional linear function where W is the weight matrix and b the bias[6] or regularization term. d_{in} stands for input dimension while d_{out} stands for output dimension. In the past decade, linear and log-linear models were the dominant approaches in NLP. In contrast, feed-forward neural networks with hidden layers are universal approximators, since they are capable of representing basically any Borel-measurable function [15].

DL models are seen as "universal approximators" because their goal is to return a function that maps the inputs to their labels as accurately as possible. To achieve this, the parameters (W and b) are set in order to minimize the loss function over the training set and the validation set [15].

4.2 Neural Network architectures

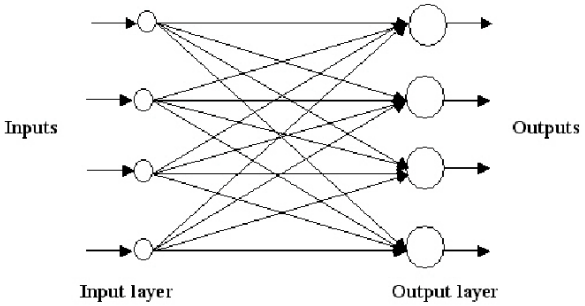

Figure 4.3: Single Layer Neural Network

In a single-layer neural network, the inputs are multiplied by the respective weights and a bias is added, this structure represents the layer of neurons. Each neuron, thanks to its activation function decides whether to fire an output or not [35]. All the firing neurons finally send their results to an output layer that gives the prediction of the overall network.

What happens when we use multiple layers instead of one? Suppose that you receive the task to develop a system that identifies images of fruits [35]. You have some images

[5]The main benefit is that it is easy to interpret.
[6]It is initialized randomly as well.

of pineapples and bananas and you develop a basic logic that can help your model identify an image based on the color and the shape. Suddenly you realize that some of the images that you need to identify are in black and white [35]. Now you need to start your coding from scratch. Some images are too complex and time consuming for a human to code, even though your brain has no problems in linking an image to the category "fruit". When such complex issues are encountered it is the time to use a DNN [35] thanks to their ability to generalize the input they see.

This surprisingly happens by stacking together multiple hidden layers in a neural network.

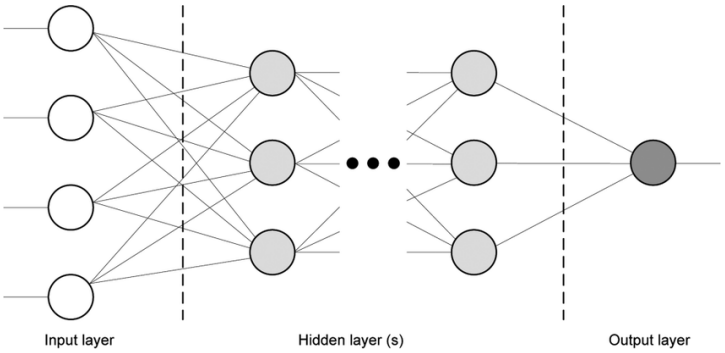

Figure 4.4: Multi-Layer Neural Network

If the DNN needs to learn to recognize a face, for example, it will first learn how to recognize its various components, such as the eyes, nose and so on [35]. Therefore, features of a higher level are derived from the lowest ones in a hierarchical process. This hierarchical approach mirrors our brain's ability to perform actions through different levels of abstraction in every situation [35].

There is evidence that the deeper[7] the neural network, the greater is its ability to perform abstract generalizations while the wider[8] it is, the more the network is able to memorize data [8]. Now we have seen a lot of machine learning concepts, but we still need to solve one important question: How can a our model learn?

4.3 Neural Network Optimization

Suppose you are a blind man on top of mount Everest, and you want to reach your town at the bottom of it. You obviously have no idea in what direction you should start walking. In this case, you feel the land nearby and try to find the way where the land tends to descend, step by step [35]. If you take your first steps in the descending direction

[7]The more hidden layer are added, the deeper the NN.
[8]The more neurons per hidden layer are added, the wider the neural network.

and each time you repeat the process, it is very likely that in the end you would have accomplished your goal by arriving at the bottom [35].

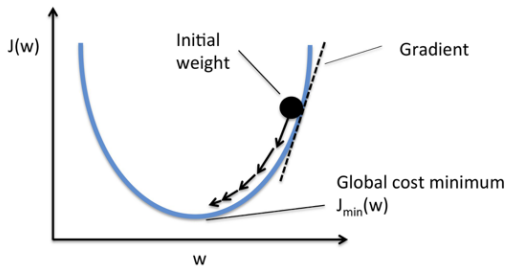

Figure 4.5: Gradient descent [35]

More formally: Gradient descent is an optimization method that we implement to minimize the error function of our model through a first-order iterative algorithm. Therefore the **gradient**, in its simplest form (with only one variable), uses the tangent line of the objective function to find the right direction we need to move towards. For sake of completeness, it does not tell us the minima right away, but shows us the proper direction we need to follow [35]. This will help us find the smallest possible error called minimum [35].

To be precise, in high-dimensional problems we are usually dealing with multiple variables and there could be many minima points (called local minima) [35]. The lowest one, like shown in Figure 4.5, is known as the **global minimum**.

Remark. Gradient is sometimes also called slope in linear regression, but this is not the same concept of the m slope we have previously encountered so do not be confused.

The equations of gradient descent are nothing but a partial derivative of the error function, which should be differentiable. Now, what is the best way to compute the partial derivative of a high dimensional problem, such as in the case of a multi-layer NN?

4.3.1 Backpropagation

To train the model we use the backpropagation procedure, short for "backward propagation of errors", whose goal is to minimize the loss function seen before [35]. Backpropagation is just an efficient method of computing the gradients in a NN environment, especially in a MLP context. This can be achieved by making wide use of the **chain rule** of derivatives, whose purpose is to differentiate composite functions, to the error function [35]:

$$\frac{d}{dx}\Big[f\Big(g(x)\Big)\Big] = f'\Big(g(x)\Big)g'(x)$$

In Layman's term, this is what happens in a NN (assuming a linear activation function for simplicity):

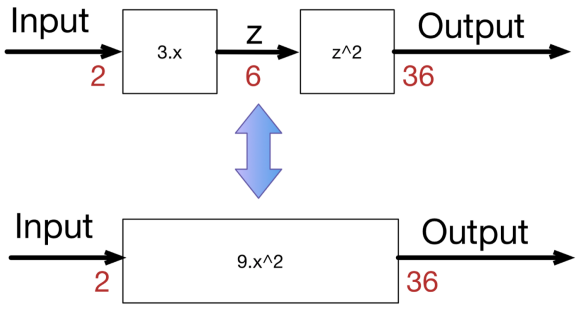

Figure 4.6: Forward Propagation [24]

The starting point is **forward propagation** [24], that consists in taking the input, which in the case of Figure 4.6 is 2, and to pass it through the NN until we see the output 36. Luckily the derivative, by being differentiable, can be now back-propagated through a decomposition, thanks to the application of the chain rule [24]. Once:

1. The error function is known.
2. The derivation of each function from the composition is also known.

At this stage it is possible to let the error flow back from the end to the beginning [24]. For instance a delta-change of 0.001 in the input, becomes a 0.003 delta-change after the first layer, which ultimately translates into a 0.000009 delta-change on the output [24]. Graphically, this is the case when we merge the two functions in the bottom part of Figure 4.6. Similarly an error on the output of 0.000009, can be backpropagated to an error of 0.003 in the middle hidden stage, then to 0.001 on the input [24].

This will allow the model to "remember the way back" by keeping track of all the differentiable layers [24]. This technique is known as automatic differentiation, and simply requires that every function has the implementation of its derivative [24]. This way of computing gradients is particularly useful, since it allows the network "to learn" in which direction the weights should be tweaked at every step [35].

Now any layer can forward its results to the next ones, in this case we sum the deltas coming from all the target layers. Thus, our calculation stack can become a complex calculation graph. Figure 4.7 shows the backpropagation process in its entirety [24].

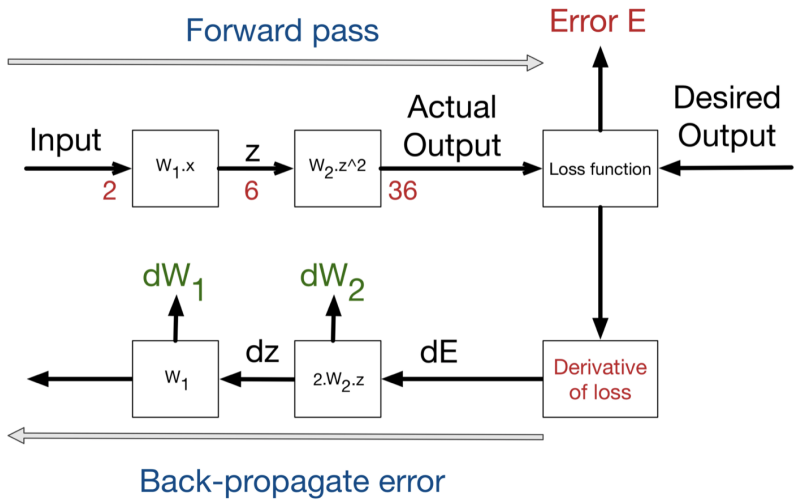

Figure 4.7: Forward Propagation [24]

For sake of completeness, the independent variables in a NN can be a lot. For this reason, in neural networks the gradient is the multi-variable generation of the derivatives, whose symbol is the nabla ∇ [35]:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix}$$

The gradient of a function f, if evaluated at multiple input (x, y, ...) points is the direction of steepest descent.

4.3.2 Learning Rate

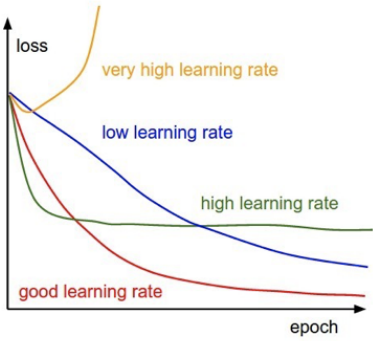

Figure 4.8: Learning rate [35]

At each iteration, the Learning rate plays a major role by showing how fast the algorithm should learn and therefore how big the "steps down the mountain" should be. In fact, if its value is too high, the model may not train at all, while if it is too low, the training time would increase exponentially. Note that it is usually represented by the symbol η.

After having an overview of how gradient descent works, now we are going to analyze the most popular versions of it [35].

4.3.3 Basic gradient descent

The simplest form of gradient descent calculates the gradient of loss function with regards to the parameters available in the whole training dataset. For a single update, the whole training dataset and every single parameters are considered, which is why this method is particularly slow [35]:

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

Where $\nabla_\theta J(\theta)$ is the gradient of the loss function $J(\theta)$ taken into account.

4.3.4 Stochastic gradient descent

In the stochastic gradient descent (SGD) the parameters are updated for each training example and label, this extremely increases the learning speed [35].

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Maybe...too fast. In fact, this model tends to overshoot the global minimum and keeps running towards a local one like shown in Figure 4.9.

4.3.5 Momentum Gradient descent

To solve the SGD problem of oscillation, we will use momentum method, which is a concept that takes inspiration from physics.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

By taking a closer look to the equation, we can notice that we are adding part of the previous time step γv_{t-1} to the current one. By doing so, the performance of the updating process (hopefully in the right direction) is substantially increased, the oscillations are reduced, and we can converge to the minimum more efficiently. However, in this case, the risk is that this algorithm, like shown in Figure 4.9, can miss the global minimum completely [35].

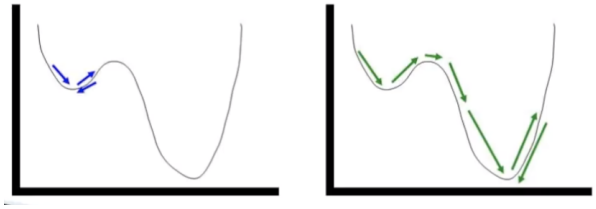

Figure 4.9: Local minimum and global minimum [35]

4.3.6 Nesterov Accelerated Gradient

This method was invented by Yurii Nesterov, who has tried to solve the problem that occurred in the momentum gradient descent by providing more dynamic parameter values.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

As you can see, this method takes inspiration from Momentum Gradient Descent, but corrects its downsides by [35]:

1. Firstly, we make a jump based like in the previous momentum case.
2. Secondly, we compute the gradient.
3. Lastly, we add a correction to generate the final update for our parameter. This helps us provide parameter values more dynamically.

4.3.7 Adaptive momentum estimation

Adaptive momentum estimation (Adam) is currently the one of best optimization method out there and it became extremely popular in the DL community. This method finds individual learning rates for each parameter by using the power of adaptive learning rates methods [35]. It uses the squared gradients to scale the learning rate and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum [7].

Adam takes its name because of its wide use of the estimations of the first and second moments of the gradient; this allows him to adapt the learning rate for each weight within the NN [7]. What exactly is a moment?

$$m_n = E\left[X^n\right]$$

The first moment is mean, and the second moment is uncentered variance[9] [7]. The gradient of the error function of a NN can be seen as a random variable (RV), given the

[9]because during variance computation the mean is not subtracted

fact that it is usually evaluated on some small random portion of data called "batch" [7]. To find the moments, Adam uses the exponentially moving averages, calculated on the gradient computed on every batch [7]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2$$

Where m and v are the moving averages, g is the gradient on a specific mini-batch, and betas are the new introduced hyper-parameters of the algorithm which represent its decay rate for respectively the first and second moment [7]; their default values are usually 0.9 and 0.999 and they are commonly left untouched. Since m and v are estimates of first and second moments, we want them to respect [7]:

$$E\left[m_t\right] = E\left[g_t\right]$$

$$E\left[v_t\right] = E\left[g_t^2\right]$$

If these properties held true, that would mean, that we have unbiased estimators. However, this is not true for because the estimators are biased towards zero given the fact that they are initialized with that number [7].

Let's prove that for m[10] by finding its pattern [7]:

$$\begin{gathered} m_0 = 0 \\ m_1 = \beta_1 m_0 + (1 - \beta_1)\, g_1 = (1 - \beta_1)\, g_1 \\ m_2 = \beta_1 m_1 + (1 - \beta_1)\, g_2 = \beta_1 (1 - \beta_1)\, g_1 + (1 - \beta_1)\, g_2 \\ m_3 = \beta_1 m_2 + (1 - \beta_1)\, g_3 = \beta_1^2 (1 - \beta_1)\, g_1 + \beta_1 (1 - \beta_1)\, g_2 + (1 - \beta_1)\, g_3 \end{gathered}$$

The more we expand m, the less first values of gradients contribute to the overall value since as they get multiplied by iteratively tinier betas. Generally, we are facing [7]:

$$m_t = (1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i$$

Thus, the expected value of m in our case is [7]:

$$\begin{gathered} E\left[m_t\right] = E\left[(1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i\right] \\ = E\left[g_i\right] (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} + \zeta \\ = E\left[g_i\right] (1 - \beta_1^t) + \zeta \end{gathered}$$

g[i] is approximated with g[t] and thus can be taken it out of the sum, since it does not now depend on i anymore. Given this approximation the error ζ appears. As you can notice we are facing a biased estimator, after correcting it we obtain [7]:

[10] the proof for v would be the same.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

After having computed the unbiased first and second moment of the gradient, we can finally use them to update the parameters [35] by scaling the learning rate for each parameter individually:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

Note that ϵ is a small number whose purpose is to avoid dividing by zero [7].

4.4 Activation functions

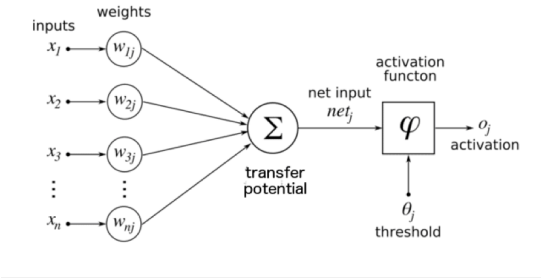

Figure 4.10: Activation Function [35]

Previously, we said that a neuron uses a step-function, the simplest form of activation function [36]. However, there are some drawbacks in using it:

Suppose you are creating a binary classifier that says "activate" or "not activate". A Step function gets the job done pretty well by saying one or zero. What about multi-class classification? sounds complicated. Intermediate activation values may give us a smoother learning process (less wiggly) [36]. Thus, what we need is an activation functions that can produce intermediate (analog) activation values rather than simply ones and zeros. The immediate solution could be:

4.4.1 Linear activation function

The linear activation function $A = cx$ is a straight line where the activation is proportional to its input. In this case if more than one neurons fires, we can take the max (or

softmax[11]) and decide what to do. What is it wrong with this approach? The problem comes when we compute the gradients during backpropagation, because the derivative, c, is a constant value [36].

The result is that the gradient has no relationship with X therefore, when the error is backpropagated, the changes do not depend on a variation of the input value x. A combination of linear functions is still another linear function [36]. Let's break this down with an example, suppose that you want to stack many layers in your NN [36]:
layer1(x) = x + 1
layer2(x) = 10x + 10
layer1(layers(x)) = (10x + 10) + 1 = 10x + 11

Did you see it? If the activation functions are all linear in nature, the final output will also be a linear function of the input of first layer! This means that, no matter how many layers get stacked together, they can be replaced by a single one [36].

The reason why we prefer using non-linear ones is that, without the non-linearity property, the NN cannot generate the complex behaviors we want to obtain [35].

4.4.2 Sigmoid activation function

The Sigmoid was one of the most popular types of activation functions when ANNs stated to became popular, its equation is:

$$g(z) = \frac{1}{1 + e^{-z}}$$

With:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & z < 0 \end{cases}$$

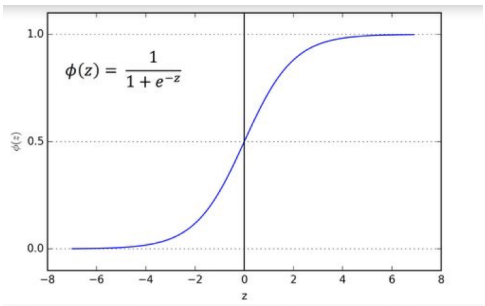

Figure 4.11: Sigmoid function [35]

[11] We will go over this function in a few pages.

Its gradient is:

$$g'(z) = \frac{1}{1+e^{-z}}(1 - \frac{1}{1+e^{-z}})$$

The sigmoid is smooth and its behavior is similar to the one of a step-function [36]. Its benefits are:

1. It is non-linear in nature [36].
2. Many layers can be stuck without the drawbacks seen for the linear activation function [36].
3. It has a smooth gradient [36].
4. It is not limited to binary classification tasks [36].
5. produces a s-shaped curve that squashes the input numbers between 0 and 1, which is way smaller than (-inf, inf) of the linear function [36].

Despite being one of the most popular activation functions for years, this method has some limits too [36]:

Vanishing gradient problem

When the activation reach near the "near-horizontal" parts of the curve on either sides, the Y values tend to change very little to any change in X [36]. This means that the gradient, in such cases "gets stuck" or "vanishes" [36]. In Layman's terms, it stops learning (or converges[12] extremely slowly) and the situation worsens the more layers are added in the model [35]. Therefore, leaves a space in which large regions of input are mapped to a very small range can lead to big risk understatements because, during training, even a huge input change may lead to a tiny output one [35].

Non-zero-centric-function

The sigmoid function is not a zero-centric activation function, this is clear from the fact that its output range is [0,1], which means and therefore it does not spin around zero [36]. This element complicates the optimization process since the gradient update can go way too far into a positive or negative direction [35].

4.4.3 Hyperbolic tangent activation function

The hyperbolic tangent function (TanH) comes to the rescue [36]. Its formula is:

$$f(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1+e^{-2x}} - 1$$

[12] learns

Looks familiar right? It is! In fact, we are looking at a scaled version of the sigmoid activation function [36]:

$$tanh(x) = 2\text{sigmoid}(2x) - 1$$

Its gradient is:

$$f'(x) = 1 - tanh^2(x)$$

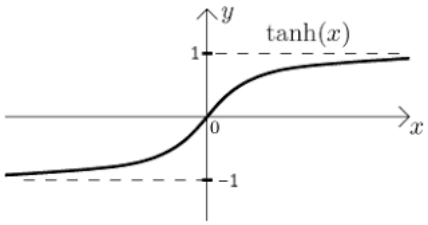

Figure 4.12: TanH activation function

Apart from all the benefits enumerated for the sigmoid function, tanh is zero-centric since its input region lays in the range [-1, 1], which makes optimization procedure easier [36]. Even though the gradient of this function is way stronger than the sigmoid, the vanishing gradient problem persists, so other solutions are necessary [35].

4.4.4 Rectified Linear Unit and its variants

Rectified Linear Unit (ReLu) is currently the most widely used activation function in this landscape [36].

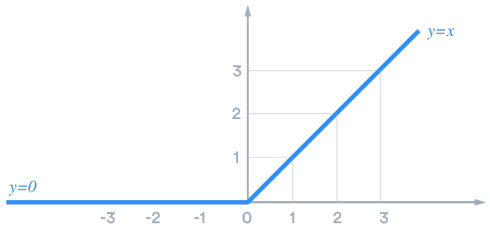

Figure 4.13: ReLu activation function

$$f(x) = \max(0, x)$$

Whose derivative is:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & x < 0 \end{cases}$$

What about f'(0)? To be precise is not differentiable, because the left derivative is zero and the right derivative is one, but it is typically defined as equal to zero [30].

From a pure theoretical perspective ReLu should not be an eligible activation function in DL algorithms. However in practice, gradient descent performs really well for ReLu to be implemented [30]. There are multiple reasons for this [30]:

1. ANN do not always reach the local minimum of the cost function, thus it is acceptable for the cost function's minima to have undefined gradient in a small number of specific points [30].

2. A computer, usually return one of the one-sided derivatives rather than reporting that the derivative is not defined or raising an error. In fact, when a machine needs to compute g(0), it has an intrinsic error for which it is very unlikely that its underlying value truly has the exact value zero. Instead, it is possible that it has a small value that is rounded to zero. This may be an heuristical justification [30].

In some contexts not strictly related to neural networks there are indeed some more theoretically pleasing justifications, but we will not dive into them since they diverge from our purposes. The main concept to take out from this discussion is that in practice we can ignore the non-differentiability of the hidden layers' activation functions in that specific case [30].

At first, since it is linear in the positive axis, you may think that ReLu would have the same issues of its linear cousin, as it is linear in positive axis; However, ReLu has nonlinear behaviors and therefore not only we can stack together layers, but we can do it with a much more flexible range which is [0, inf) [36].

Another important matter is the activation's sparsity: Imagine having a big NN with a lot of neurons, with sigmoid or tanh almost all neurons will be processed to produce the network's output[13] [36]. This means that we are dealing with a dense activation that, depending on your problem, is computationally expensive. ReLu comes to the rescue by reducing the number of non-necessary neurons that fire (sparse activation), the result is a faster and lighter model. ReLu works most of the time as a general approximator, but may sometimes encounter one major problem [36].

What about the horizontal line for negative X? The gradient of that region gets stuck at zero, therefore the neurons that go into that state will "die" by becoming insensitive to any input/error change [36]. This why we address this issue as "dying ReLu problem". To overcome this limitation, Leaky ReLu (Figure 4.14) has been created by transforming the horizontal line into a non-horizontal element. The intuition is the following: letting the gradient be non-zero so that it can (eventually) be useful again [36].

$$f(x) = \begin{cases} \alpha x \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$$

This solves the issue of the neurons' death, but with the cost of slowing down the training process. In conclusion, the ReLu function should be used, but if you notice the

[13] To use a technical term: all neurons fire in an analog way.

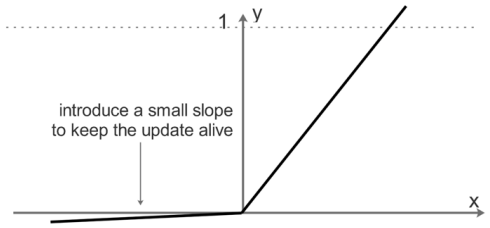

Figure 4.14: Leaky ReLu activation function

issue of those weakened neurons, Leaky ReLu could be a better solution [35].

Parametric ReLu (PReLu) is an activation function that pushes the concept of Leaky ReLu even further: usually Leaky ReLu has a α value fixed at 0.01, while PReLu lets it free to be modified by the neural network and, therefore, flexible.

4.4.5 Exponential linear unit

Batch Normalization

One of the issues in Artificial Neural Networks is the so called covariance shift (or drifting) [17]. Let's make an example to have a better intuition of what is going on: Suppose having to train your model on a dataset containing only individuals between 18-30 years old, while the validation set contains data relative to much older individuals [17]. This means that the distribution between training and testing datasets differs significantly and our machine might have a high chance of failing. DL algorithms ignore these changes even when they should not [17].

Batch normalization is a technique for improving the performance and stability of NNs. It can be thought as applying some form of preprocessing at every layer level [12]. It reduces this risk by allowing each NN layer to be more flexible and learn from one another more smoothly [12].

How is normalization useful and seems to be so effective [12]? Suppose a NN has some values concentrated between 6 to 55 and some others from 100 to 500. Normalization helps the NN have a common input interval and therefore to be more stable [12].

More formally, given a mapping from X to Y learned by a DNN: if a change in the x distribution comes up, we might want the NN to be able to realign the distribution of X with the distribution of Y without the need to retrain the model every time from scratch [12]. Batch-Normalization achieves exactly this goal by adjusting and scaling the activation function of each hidden layer. This is possible simply by adding a Batch-Normalization (BN) layer before each activation function [11]. Mathematically, a this is

how a BN layer works:

Mini-batch mean:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

Mini-batch variance [12]:

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$

Normalize [12]:

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

Scale and shift through batch normalization [12]:

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

Like leaky-ReLu and PReLu, the exponential linear unit (ELU) shown in Figure 4.15 speeds up the learning process in the DNN and shows better classification performance than its ReLu predecessor [11] by getting rid of the vanishing gradient problem. This is due to the fact that ELU accepts negative values and therefore allows the mean unit activations to be pushed to zero and thus brings the gradient closer to the unit natural gradient [11]. The mathematical formula is:

$$f(x) = \begin{cases} \alpha\,(e^x - 1) \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$$

while its derivative:

$$f'(x) = \begin{cases} f(x) + \alpha \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$$

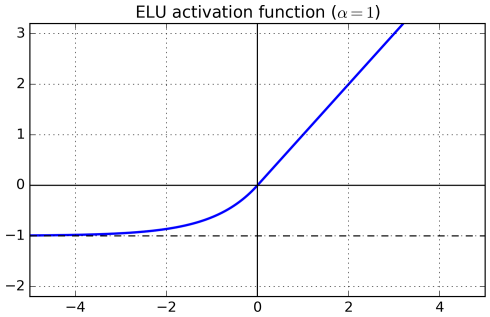

Figure 4.15: Exponential linear unit with $\alpha = 1$ [20]

ELU, thanks to its structure, implements the benefit presented in the section of Batch Normalization automatically and therefore with a significantly smaller computational cost [11]. While other activation functions such as Leaky-ReLu and PReLu present negative values too, they do not ensure the same noise-robust deactivation state that ELU can be proud of [11].

4.4.6 Softmax function

The softmax function is used for giving output vectors a probability form [35]. The kth neuron's output will be calculated by:

$$y_k = P(w|c) = \frac{exp(h^T v'_w)}{\sum_{w_i \in V} exp(h^T v'_{w_i})}$$

Where h is the output vector of the penultimate network layer, c is the context, w_t is the target word, w_i is the embedding of every word in the vocabulary V. To generate an error vector, we need to subtract the probability vector from the target vector, and once we know the error vector, we can fine tune it by adjusting the weights [35].

4.5 Neural Networks Types

4.5.1 Feed-forward Neural Networks

In a feed-forward neural network the work-flow is simple, the data goes from the input layer[14],to the hidden layers until the output layer; therefore the information can go only in one direction...forward [15].

"Even though the brain's metaphor is intriguing, it is also distracting and cumbersome to manipulate mathematically. A feed-forward neural network is simply a stack of linear models separated by activation functions with nonlinear behaviors. In which the values of each row of neurons in the network can be thought of as a vector." [15]. The form of a feed-forward NN with one hidden layer is:

$$NN_{\mathrm{MLP}_1(x)} = g(xW^1 + b^1)W^2 + b^2$$
$$x \in R^{d_{in}}, W^1 \in R^{d_{in} x d_1}, b^1 \in R^{d_1}, W^2 \in R^{d_1 x d_2}, b^2 \in R^{d_2}$$

Where W^1 and b^1 are the terms for the linear transformation of the input x from d_{in} to d_1. g is then applied to each of the d1 dimensions, and the matrix W^2 together with bias vector b^2 are then used to transform the result into the d_2 dimensional output vector. Without the non-linearity property in g, the neural network can only represent linear transformations of the input, which would make all of these efforts useless [15]. Mathematically, a NN with two hidden layers (represented with the letter h) can be written as:

$$\begin{aligned} NN_{MLP_2}(x) &= y \\ h^1 &= g^1(xW^1 + b^1) \\ h^2 &= g^2(h^1 W^2 + b^2) \\ y &= h^2 W^3 \end{aligned}$$

[14]Which is the first "column" of neurons

A layer is simply a vector resulting from each linear transformation that will expect a d_{in} vector as input and transform it into a d_{out} output vector.

4.5.2 Recurrent Neural Network Architectures

Our thoughts have persistence. When we think, we do not forget everything we experience and start our thoughts from zero every second. As you read this thesis, you are comprehending every section based on what you understood of the previous ones [27]. Traditional feed forward neural networks are not built to act precisely in this way, in fact, it seems like a big inefficiency. Imagine, for instance, that you want to classify what kind of event is happening at every point in a film. It's unclear how a traditional NN could accomplish this task [27].

Recurrent neural networks have been created exactly to address this issue. They are networks characterized by having loops, that allow such information to remain alive within the NN memory [27]. This is how we are going to proceed:

- Firstly we will introduce a general description of what a Recurrent neural network (RNN) is.
- Then, we will go deeper in the more complex gated structures such as LSTM.

4.5.3 Recurrent Neural Networks

Simple recurrent neural networks

Recurrent neural networks (RNN) use sequences as inputs in fixed-size vectors, rather than ignoring the sequential nature of data like some of its predecessors. The simplest RNN algorithm that is sensitive to the ordering of elements in the sequence is known as an Elman Network or Simple-RNN (SRNN) [15]. Mathematically and graphically:

$$s_i = R_{SRNN}(x_i, s_{i-1}) = g(s_{i-1}W^s + x_iW^x + b)$$
$$h_i = O_{SRNN}(s_i) = s_i$$

$$x_i \in R^{d_x}, h_i, s_i \in R^{d_s}, W^x \in R^{d_x x d_s}, W^s \in R^{d_s x d_s}, b \in R^{d_s}$$

In Figure 4.16, a chunk of NN called A, looks at some input x and outputs h. A loop allows information to be passed from one step of the network to the next [27]. This means that the hidden state s_{i1} and the input x_i are each linearly transformed, the results are added (together with a bias term) and then passed through an activation function g. The output at position[15] i is the same as the hidden state in that position [15].

[15]position, in this case, can be considered as "time".

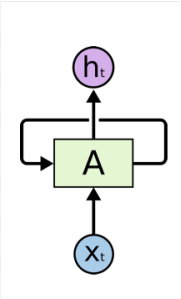

Figure 4.16: RNN Scheme [27]

The presence of a non-linear activation function makes the network order-sensitive. This is why RNNs became so popular so quickly: they are very powerful at capturing patterns in sequential input s [15]. The RNNs are defined recursively, which means that a function R, taking as input a state vector s_{i-1} and an input vector x_i and returning a new state vector s_i.

The state vector s_i is then mapped to an output vector h_i using a simple deterministic function O(·). The initial input of the recursion is an initial state vector s_0[16].

Probably Figure 4.16 could be more easily understandable if the recursion would be "unrolled" to explicitly clarify what we mean by "state s_{i-1}, s_{i_1} etc.", resulting in the Figure 4.17. *"A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor"* [27].

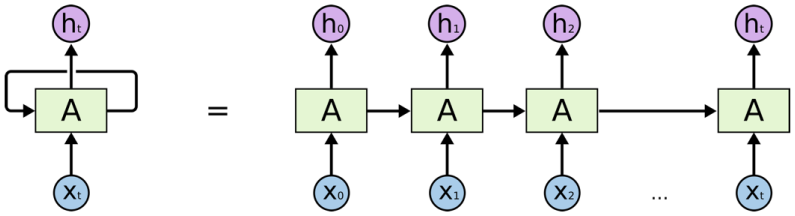

Figure 4.17: RNN unrolled [27]

To train a RNN, then, all we need to do is to create the unrolled computation graph for an input sequence, add a loss, and then use the backpropagation procedure that has been previously introduced [15]. Why are we putting so much effort in training a single cell?

We can visualize the different neuron architectures as "Lego Bricks" that we can use to create more complex structures to help us achieve certain behaviors [15]. In the

[16]For simplicity, we omit the initial vector s_0 by assuming it to be the zero vector.

sentiment analysis case the RNN can be seen as an acceptor: We observe the final state, and then decide on an outcome. In other words, a RNN reads each word/sentence in a document and, depending on the final state of the output layer, it decides if the overall document portrays a positive or negative sentiment [15].

4.5.4 Gated Architectures

As we have seen, the biggest strength of RNNs is that they may be able to link past data for solving current problems. However, can they always do it? It depends [27].

Sometimes, recent information are the only needed, in case like in "this American girl lives in the USA" it is straightforward that the next word is going to be USA, no further context is necessary [27]. Whenever the gap between the relevant information and the linked concept is small, RNNs can learn to use the past data easily. However sometimes we may need much more context which translates in a wider gap, consider for instance: "We were born in Italy...[a few lines]...we speak Italian" [27].

Unfortunately as this gap grows S-RNN structures may fail. The closely related word "speak" suggests that the next word is most likely the name of a language, but if we want to forecast which language, we need the context of "France" from further back [27].

This drawback of the S-RNN is that it is difficult to train properly because the vanishing gradients problem is particularly strong. Gating-based architectures have been created to overcome this issue [15]. How do we achieve that? Consider a binary vector $g \in 0, 1^n$. Its purpose is acting like a gate system, that we will call "hard-gating mechanism" for controlling the access to the R^n vectors, using $x \odot g$ [15].

Consider a memory $s \in d$, an input $x \in R^d$ and a gate $g \in (0, 1)^d$. The computation $s \longleftarrow g \odot x + (1 - g) \odot s$ "reads" the entries in x corresponding to 1 in g, and rewrites them in s [15]. Perhaps it will be clearer through a graphical representation like in Figure 4.18.

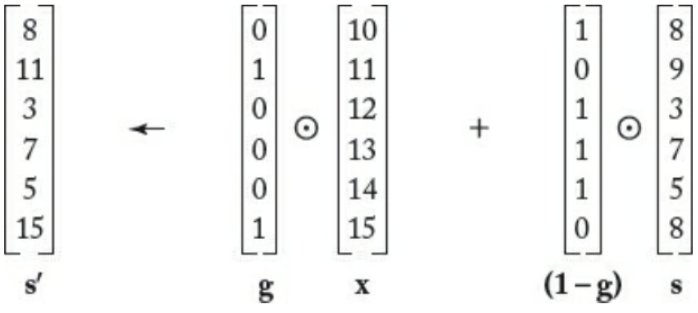

Figure 4.18: Gating example [15]

The gating mechanism described above can serve as a building block in our RNN: gate vectors can be used to control access to the memory state s^i. However, we are still missing two important (and related) components [15]:

- the gates should not be static.
- their behavior should be learned.

A solution to this obstacle is to substitute the so-called "hard-gating mechanism" we have just encountered with a "soft-gating mechanism" whose main difference is being differentiable [15]. To do so, we replace the requirement of g being $g \in 0, 1^n$ and allow arbitrary real numbers, $g \in n$, which are then passed through a sigmoid function $\sigma(g)$ [15].

This bounds the value in the range (0, 1), with most values near the borders. When using the gate $\sigma(g) \odot x$, indices in x corresponding to near-one values in $\sigma(g)$ are allowed to pass, while those corresponding to near-zero values are blocked. The gate values can be conditioned on the input, memory and be trained through backpropagation. The result is a flexible model where at each time step, this differentiable gating mechanism decides which parts of the inputs should be memorized or not [15]. Now we will see some technical implementations of these concept by analyzing the most widely adopted gated-neurons.

4.5.5 Long-Short Term Memory

Long Short Term Memory (LSTM) NNs are a special kind of RNNs which are specifically designed to easily avoid the long-term dependency problem of the vanishing gradient thanks to the "soft-gating mechanism" [27].

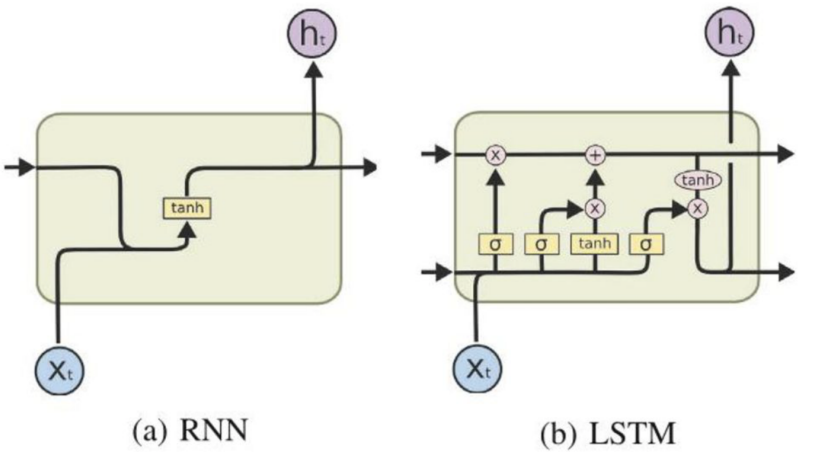

Figure 4.19: RNN (a) vs LSTM (b) [15]

As we have previously said, all RNN are characterized by a having a loop that, when unfolded, has the form of a chain of repeating modules of NNs [27]. In S-RNNs, this modular chain is quite simple, in fact it only contains a a single tanh activation layer like in Figure 4.19 (a). In LSTM, these architectures are way more complex, like in Figure 4.19 (b). Before moving on we need to clarify some symbolism:

Figure 4.20: Important symbols [27]

An LSTM is divided in two main parts:

1. One memory cell, whose aim is to preserve memory.
2. One working memory that takes an input and gives an output.

At every input state, the new input will be filtered by a gate that shall decide what should be forgotten. Both mathematically and graphically can be pretty scary, thus the best way to proceed is to face it step by step:

The key to LSTMs is the cell state C [27], which is represented by the horizontal line running from left (C_{t-1}) to right (C_t) at the top of Figure 4.21.

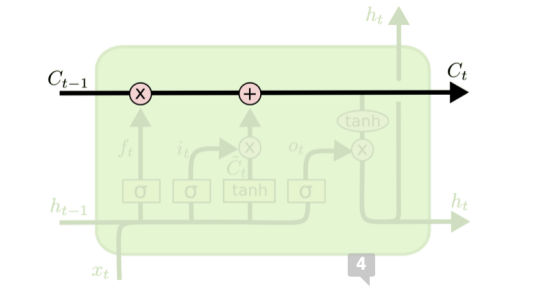

Figure 4.21: LSTM: cell state [27]

"The cell state is kind of like a conveyor belt" [27] because information easily flow down the entire chain, with only minor linear interactions with the gates. As we have stressed multiple times by now, the sigmoid layer outputs numbers [0,1], which tells us how much of each component should pass or be blocked: A 0 means "get rid of all this completely" while a 1 "keep all of this" [27].

An LSTM has three gates, used to flexibly modify the cell state:

1. Forget gate layer: regulates how much of the existing memory to forget and will be studied using the symbol "f".

2. Input gate layer: regulates how much of the new cell state to keep and will be studied using the symbol "i".

3. Output gate layer: regulates how much of the cell state should be exposed to the next layers of the network and will be studied using the symbol "o".

Firstly, the LSTM needs the "forget gate layer" to decide what data it is going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate' [27].

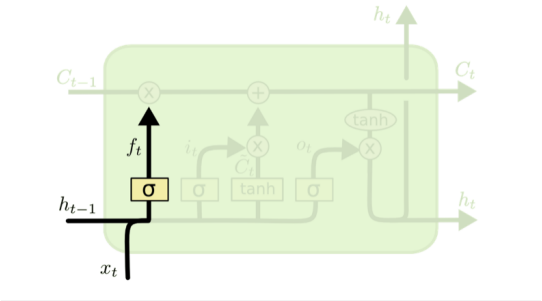

Figure 4.22: LSTM: forget gate [27]

As shown in Figure 4.22 looks at h_{t1} and x_t, and outputs a value between 0 and 1 for each number in the cell state C_{t1}.

$$f_t = \sigma(W_f \cdot [h_{t1}, x_t] + b_f)$$

In the second step, represented in Figure 4.23 we need to decide what new information needs to be kept in the cell state [27].

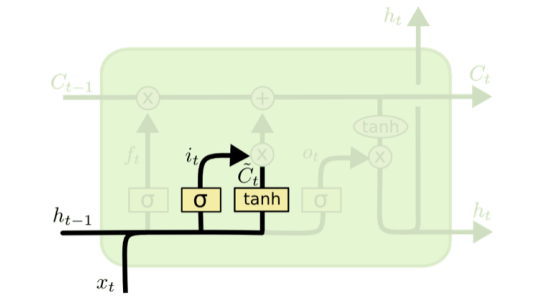

Figure 4.23: LSTM: new information to be kept [27]

$$i_t = \sigma(W_i \cdot [h_{t1}, x_t] + b_i)$$
$$\widetilde{C}_t = \tanh(W_C \cdot [h_{t1}, x_t] + b_C)$$

On the one hand, an "input gate layer" i_t is represented by a sigmoid layer which values need to be updated. On the other hand, a tanh layer creates a vector $\widetilde{C}_t$ of new possible values [27].

Consequently, these two outputs are combined thanks to a pointwise multiplication; the resulting value, shown in Figure 4.24, will be added to the old cell state C_{t-1} that has already gone through the forget gate [27].

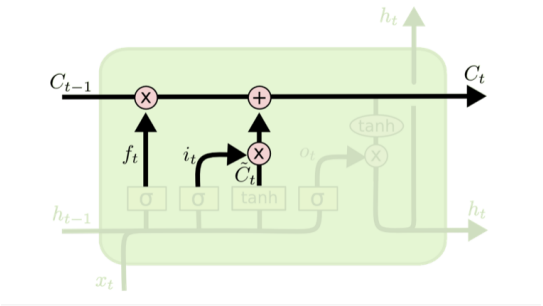

Figure 4.24: LSTM: update the old cell state [27]

This process allows the network to substitute the old forgotten information (first operation) with new one (second multiplication).

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t$$

Deciding the output, which will be a filtered version of the cell state, is the final step of our journey.

Firstly, as in Figure 4.25 we deal with a sigmoid layer which decides what parts of the cell state we need to output o_t. Consequently, the cell state is passed through a tanh layer, multiplied by the output of the sigmoid gate that we have just talked about [27].

$$o_t = \sigma(W_o \cdot [h_{t1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

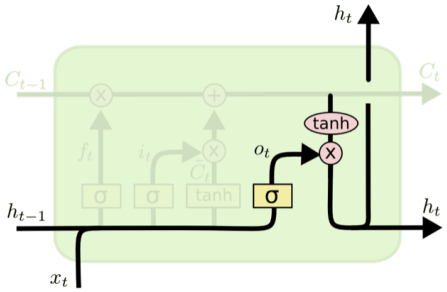

Figure 4.25: LSTM: output and new cell state [27]

Exploding gradient problem

In RNNs, error gradients can accumulate and become very large, this leads to large updates of the weights and thus instability [4]. This happens when the gradients with high values through the network layers keep growing by being consistently multiplied. Pushed too far, this issue can overflow the NN and result in NaN values [4].

To avoid this scenarios, it is common practice to use some kind of regularization method, such as Dropout.

4.6 Dropout

Multi-layer networks are usually complex and large models in which overfitting is a real problem, therefore model regularization such as dropout play a crucial role in this scenario. Their usage intuition is simple, they help the model "forgetting stuff" [15].

Dropout randomly sets to zero part of the neurons in a specific layer during the training process [15]. Mathematically:

$$
\begin{aligned}
NN_{MLP_2}(x) &= y \\
h^1 &= g^1(xW^1 + b^1) \\
m^1 &= \text{Bernoulli}(r^1) \\
\tilde{h}^1 &= m^1 \odot h^1 \\
h^2 &= g^2(\tilde{h^1}W^2 + b^2) \\
m^2 &= \text{Bernoulli}(r^2) \\
\tilde{h}^2 &= m^2 \odot h^2 \\
y &= \tilde{h^2}W^3
\end{aligned}
$$

[17] https://visualstudiomagazine.com/articles/2014/05/01/neural-network-dropout-training.aspx

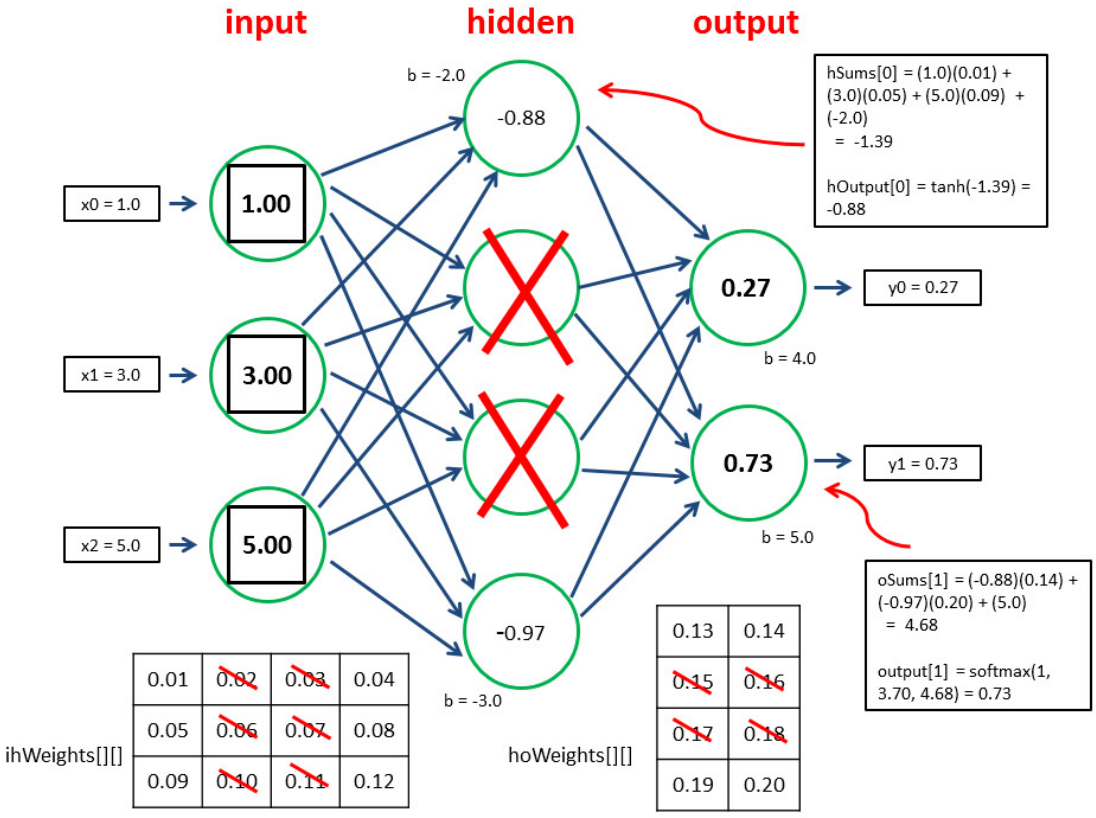

Figure 4.26: Dropout [17]

In this case, m^1 and m^2 are random masking vectors with the same dimensions of h^1 and h^2 [15].

4.7 Advanced Initialization

Random initialization, following either a normal or uniform distribution is a great way to start the weights of a NN [14]? Not really, in fact there is a whole literature on how to improve our NN's performance based on weights initialization and now, we will have an intuition of what happens behind the scenes. Note that the explanation and the images that will follow are based on a simple MLP with 5 hidden layers with 100 units each [14].

When MLP first appeared there were indeed randomly initialized weights and a simple sigmoid activation functions, however convergence was slow, training was hard, and results were unsatisfactory [14]. The common procedure was to draw random weights from a Standard Normal distribution and multiply them by a small number, the result was a small standard deviation, for example 0.01 [14].

As shown in Figure 4.27 we notice that when we start training, the activation func-

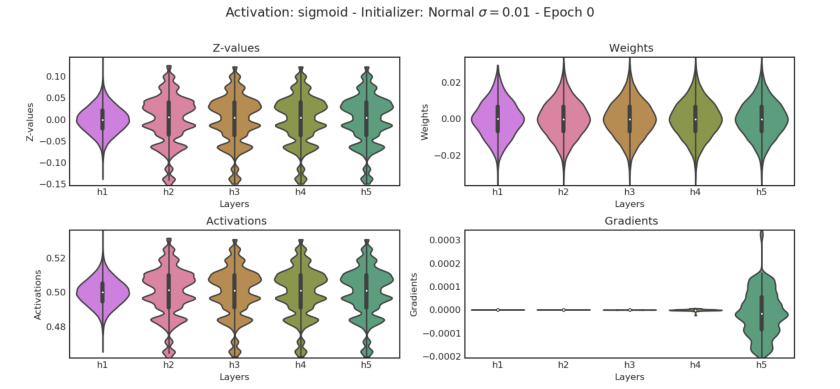

Figure 4.27: Initialization with with $\sigma = 0.01$ and sigmoid [14]

tions and z-scores[18] had a narrow range and suffered from the gradients vanishing problem, an issue that we met in Section 4.4.2. What would happen, like in Figure 4.28, if we use larger σ [14]?

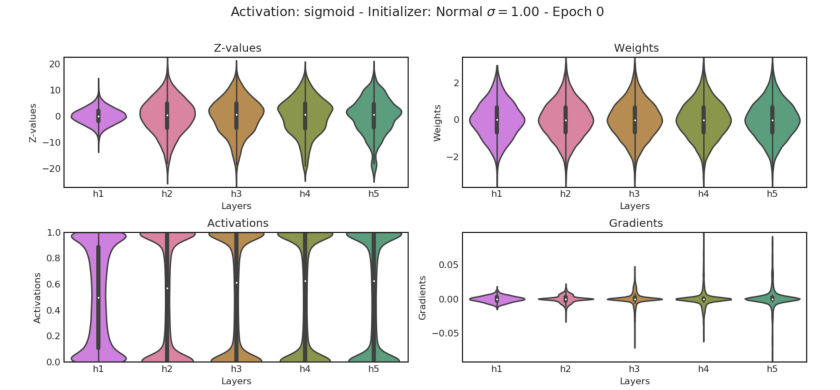

Figure 4.28: Initialization with with $\sigma = 1.00$ and sigmoid [14]

Undoubtedly the vanishing gradients problem has improved since the ranges of all layers are now much more similar. However, we notice that the range of the Z-values is too wide and forces the activation function into a binary state. How about changing the activation function all together [14]?

Using Tanh like in Figure 4.29 not only did not solve the issues of the Z-values and Activations, but we also fell on the opposite side of the spectrum with the exploding gradients problem this time. This is noticeable by seeing how the gradient values grow bigger and bigger during backpropagation [14].

[18]Remember that z-values show how many standard deviations an element is from the mean and it is calculated by [14]: $(X - \mu)/\sigma$

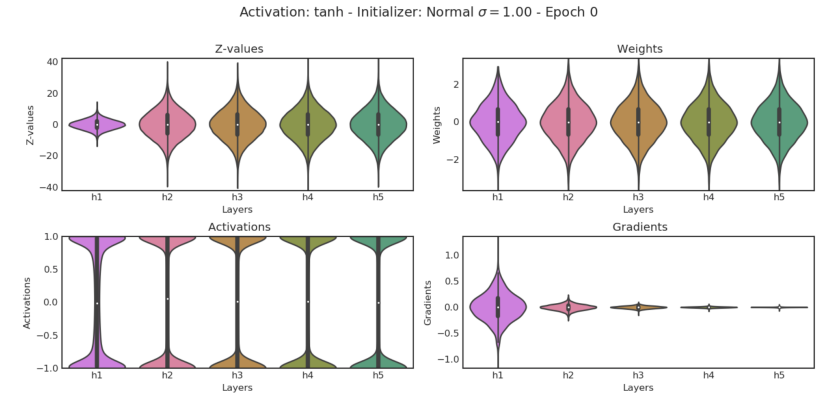

Figure 4.29: Initialization with $\sigma = 1.00$ and tanh [14]

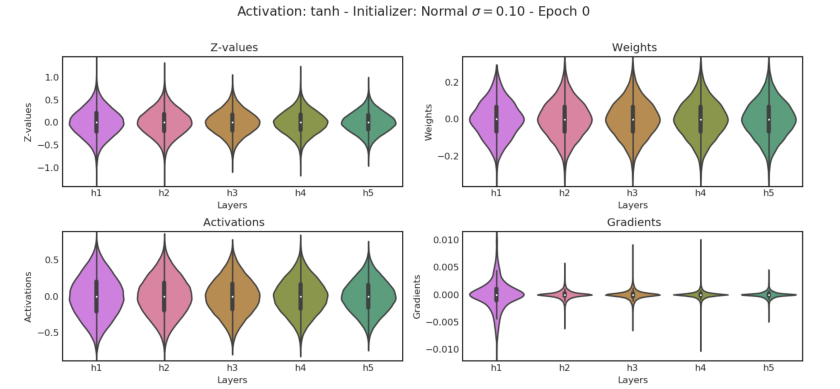

Figure 4.30: Initialization with $\sigma = 0.10$ and tanh [14]

However, tanh has the benefit of being zero-centered and, by reducing the standard deviation back to $\sigma = 0.10$ as in Figure 4.30, we finally obtain reasonably similar gradients, z-values and activations along all the layers. This allows us to stack layer after layer and expect, without surprises, similar distributions [14].

So, we need an initialization scheme that uses the best possible standard deviation for drawing random weights!

4.7.1 Glorot Initialization

But being similar along all the layers is not the cause, but the effect! Glorot Initialization (also known as Xavier Initialization) keep all the winning features listed above by keeping the variance similar along all the layers. How does it achieve that? The key is the way this approach deals with the standard deviation of the weights [14].

Their model assumes that inputs should have zero mean for this to hold in the first layer, so always scale and center your inputs [14]. Why? because if you remember some

properties of the variance:

$$\text{Var}(xW) = E[x]^2\,\text{Var}(W) + E[W]^2\,\text{Var}(x) + \text{Var}(x)\,\text{Var}(W)$$

$$E[x] = E[W] = 0 \Longrightarrow \text{Var}(xW) = \text{Var}(x)\,\text{Var}(W)$$

This is another point in favor of tanh, let's go a little deeper with a simple example [14]. Assume having only two hidden layers X and Y as shown in Figure 4.31, fully connected and with linear activation function[19] for sake of simplicity [14].

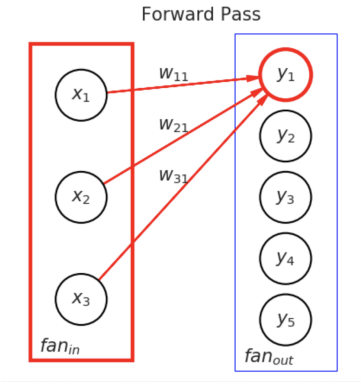

Figure 4.31: Forward pass [14]

Assuming that x and W are iid, the variance of y1 is [14]:

$$y_1 = x_1W_{11} + x_2W_{21} + x_3W_{31}$$

$$\text{Var}\,(y_1) = \sum_{i=1}^{3} \text{Var}\,(x_i)\,\text{Var}\,(W_{i1}) = n_i\,\text{Var}(x)\,\text{Var}\,(W_{i1})$$

To keep a similar σ along all the layers, we should aim for making the variance of x the same as the variance of y [14]; in our example [14]:

$$\text{Var}\,(y_1) = \text{Var}(x) \Longleftrightarrow \text{Var}\,(W_{i1}) = \frac{1}{n_i} = \frac{1}{fan_{in}}; \sigma(W) = \sqrt{\frac{1}{fan_{in}}}$$

Now to apply a similar procedure as in the forward pass to keep the variance similar along all the layers during backpropagation [14]. This time, it will be named Backward pass as represented in Figure 4.32 [14].

Thus, we obtain:

$$\text{Var}\,(\Delta_{x_1}) = n_j\,\text{Var}\,(\Delta_y)\,\text{Var}\,(W_{1j})$$

[19]This means that the activation values are equal to the z-values

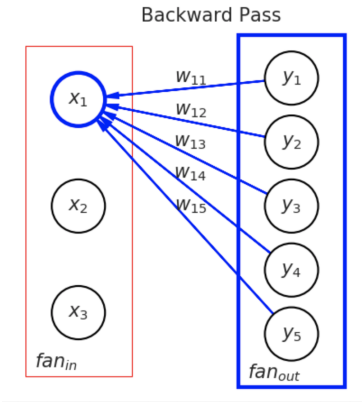

Figure 4.32: Backward pass [14]

To keep the variance of gradients similar through all the layers, the variance of its connecting weights is [14]:

$$\text{Var}\left(\Delta_{x_1}\right) = \text{Var}\left(\Delta_y\right) \Longleftrightarrow \text{Var}\left(W_{1j}\right) = \frac{1}{n_j} = \frac{1}{\text{fan}_{out}}$$

What happens if "fan-in" and "fan-out" are extremely different? Take the average [14]!

So, we finally arrive at the same expression for the variance of the weights that we find in Glorot and Bengio for the Normal distribution [14]:

$$\text{Var}(W) = \frac{1}{\frac{fan_{in}+fan_{out}}{2}} = \frac{2}{fan_{in}+fan_{out}}; \sigma(W) = \sqrt{\frac{2}{fan_{in}+fan_{out}}}$$

If we prefer a Uniform distribution [14]:

$$\text{Var}(\text{Uniform}(-\text{limit}, \text{limit}\ )) = \frac{limit^2}{3}$$

$$\text{Var}(W) = \frac{2}{fan_{in}+fan_{out}} \Longleftrightarrow \text{limit} = \sqrt{\frac{6}{fan_{in}+fan_{out}}}$$

4.8 Modeling with recurrent neural networks

RNNs can be used as acceptors, which means that they have to read an input sequence, and produce a multi-class answer at the end. This kind of operations are highly recommended since the RNNs are very strong sequence learners, and can identify very complex patterns among the datasets by working in multiple dimension cases.

Sentiment classification

In a simplistic approach, such as in a sentence-level or document-level sentiment classification task, we are given a sentence/document and need to assign it a binary value:

positive or negative. Sometimes these kind of classifications are not immediate, for instance: "It's a disappointing that it only manages to be decent instead of dead brilliant" [15] might take a while for the user to realize that it is indeed Negative.

A correct sentiment prediction requires the understanding of both the individual phrases and of the context in which they occur. The biggest difficulties in this field are, for instance, dealing with complex human expressions such as sarcasm or metaphors.

For cases with longer sentences, such as in document classification for instance, it is useful to use a hierarchical architecture, in which the sentence is split into smaller spans (or pieces) based on punctuation. The difficulty of document-level classification is that individual sentences may convey different sentiments than the one conveyed by the document as a whole [15].

4.8.1 Modelling trees with Recursive Neural Tensor Networks

As we have seen RNN is a very effective method for modeling sequences. Now I will introduce a method which is on open research in NLP: tree structures [33]. The idea behind the use of trees is that the language, the sentences, the words we use are all interconnected and linked by the grammatical structure [33].

The use of connectors such as "but", for example, can completely reverse the meaning of the initial sentence. This is why Socher [33] introduced the so called Recurrent Neural Tensor Networks (RNTN) model which, by using a tree structure is supposed to give better accuracy in more complex cases; a visual example can be seen, for instance, in Figure 4.33.

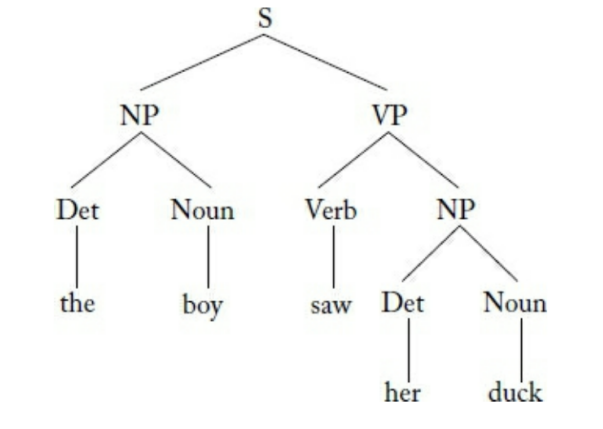

Figure 4.33: Tree representation with RNTN [15]

The training procedure for a recursive neural tensor networks is the same as before. The idea of tree-structured networks is powerful, intriguing, and seems to start from the

right intuition for dealing with the recursive nature of human language. Unfortunately, it seem that these method does not show any consistent benefit over simpler structures yet; thus further research is necessary [15].

4.9 Tuning

One of the most common questions in this field is: How do I pick the parameters' values to insert in the deep learning model? There are four common ways:

4.9.1 Babysitting

Babysitting (also known as "Trial and Error" or "Grad-Student Descent") is completely manual approach that is widely used in by researchers in various situations until they run out of time [16].

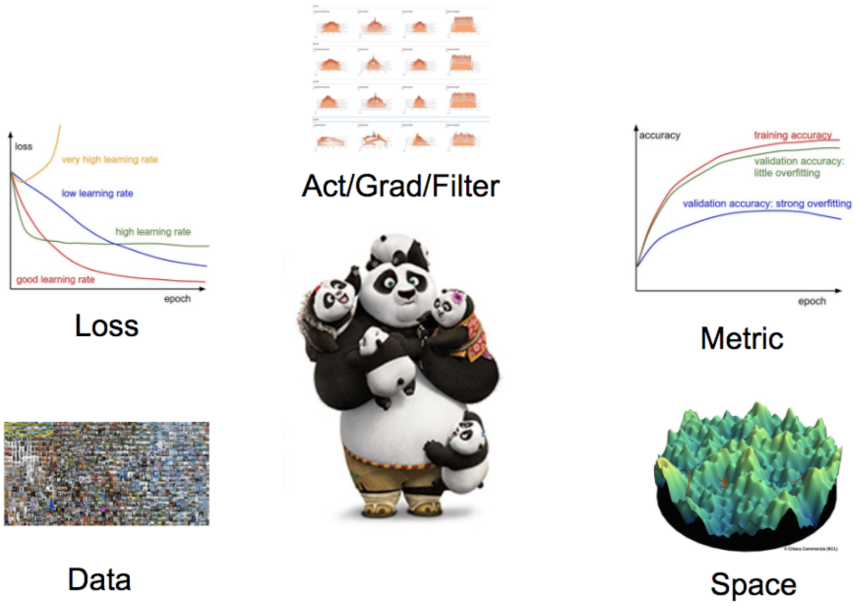

Figure 4.34: Babysitting [16]

"Can I spend my time more efficiently?" Yes! there are methods for automatizing this process and they all have pros and cons [16].

4.9.2 Grid Search

Grid Search's underlying idea is: "Just try everything!" [16]. The work-flow looks like this:

1. You define a grid of various dimensions you need. In Figure 4.35 we show Dropout rate and Learning rate for example.
2. Choose the configurations that make the most sense to you.
3. Let the computer do the job.

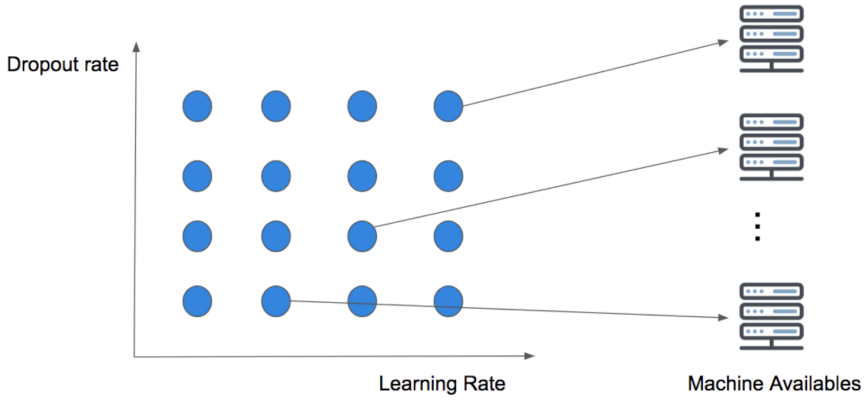

Figure 4.35: Grid Search [16]

On the one hand, the bigger your computational resources are, the more effective this technique is (especially if you can parallelize your processes). On the other hand, this method does not take into account the history of your results and it can be the more time-consuming and expensive the more dimensions you add. Typically, this method is suggested when the dimensions that you are working on are less or equal to four [16].

4.9.3 Random Search

The only real difference between Grid Search and Random Search is that the ladder chooses a random point from the configuration space. The image in Figure 4.37 gives a visual example by comparing those two approaches. In a two hyperparameters space it is assumed that one parameter is more important than the second; This is a safe assumption since DL models contain lots of hyperparameter and the researcher should have an idea of which ones are the most relevant [16].

Like in Figure 4.35 we are comparing two dimensions in order to find the best configuration [16]. In the Grid Layout, even by training nine models, we have used only three values per variable. With Random Search on the other hand, the same event is highly unlikely. With the second approach, in fact, we will have trained nine models using nine different values for each variables. By consequence, it is clear that the space

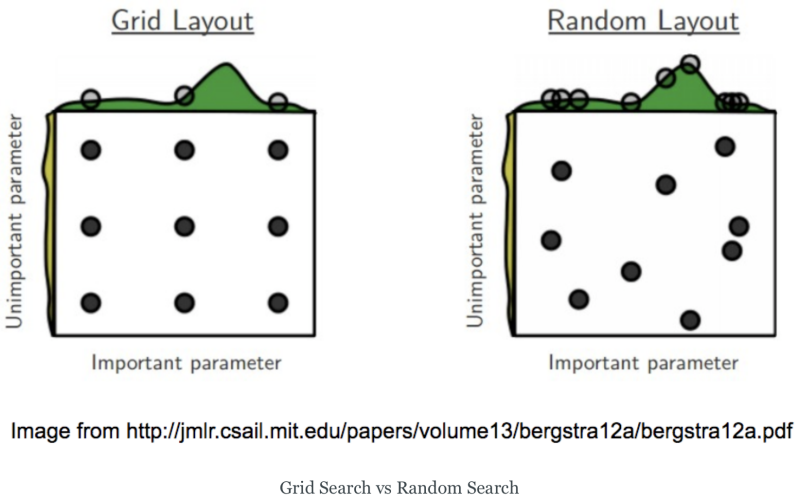

Figure 4.36: Random Search [16]

on top of Figure 4.37 (on the right) has been explored in much more detail than then one of grid search (on the left). This surely help in saving computational power and time [16].

Unfortunately, both these two methods share the same downside: "Each new guess is independent from the previous one!" [16]. Surprising, what makes Babysitting effective is the researcher's ability to to improve the next hyperparameters combination by learning from the past [16]. Would it be possible not only to automatizing the hyperparameters tuning process, but to make it smart as well? Sounds familiar? [16]

4.9.4 Bayesian Optimization

We will not go into detail of this approach since it goes beyond the scope of this research, but it still could be useful for further implementations. This search strategy consists in creating an machine learning mechanism, whose job is to find the optimal hyperparameter combination, using an iterative approach [16].

At each new iteration, the ML model will be the more and more confident about which new guess can lead to improvements [16].

4.10 Deep learning in NLP

The early era of NLP was based on the rule-based system, mainly because of the lack of both computational power and data [35]. Nowadays, DL opened the doors to statistical and probability based approaches that go far beyond what it was thinkable. DL provides in fact many benefits [35]:

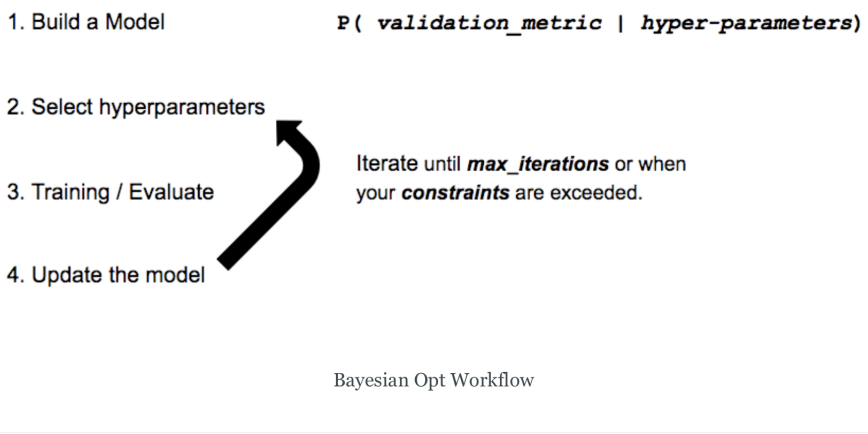

Figure 4.37: Bayesian Optimization [16]

1. Expressibility: It shows how well the machine can perform.
2. Trainability: It shows how fast a DL system is able to learn and solve a problem through a meaningful output.
3. Generalizability: This indicates how well the machine can output a result for data that has never encountered before.
4. Modularity.

NLP problems are very difficult to deal with because language itself is complex, both because it in continuous change and because is highly influenced by the context in which it happens [35]. In DL techniques, the preprocessing should be minimal, then we transform the text into vectors and finally use them as input into the DL model [15].

Neural networks are not a silver bullet for NLP problems, since the core challenge remains: language is very ambiguous, we cannot control it regardless from its context (which is exactly what NNs struggle with) [15].

To sum up, deep learning has reached great results so far, but it still has a long way to go since it does not simply have to deal with linguistics, but also with the context in which the NL is expressed and the goal the wants to achieve [15].

Chapter 5

Vectorization modeling

5.1 Basics of natural language processing features

In NLP models we consider word-tokens as our features that can be measured by counting how many times they appear in the corpus's text. We also aware that words have a structure since they are linked by grammatical rules like shown in Figure 5.1.

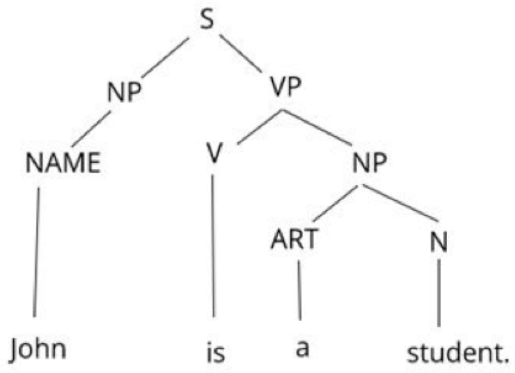

Figure 5.1: Grammar rules give a "Tree structure" to the language[1] [35]

What if an algorithm could look at words from a probabilistic perspective [23]? It makes sense, after all...a noun is usually preceded by an article and so on. We will start with the simplest approaches to this problem and we will increase the complexity more and more.

5.1.1 Bag of words model

Let's tackle our problem step by step with a simple example, consider two sentences:

[1]S means sentence, NP means noun phrase, VP means verb phrase, V means verb, N means noun, ART means article.

- Sentence 1="You may say I'm a dreamer"
- Sentence 2="We all live in a yellow submarine"

Now we will build, starting from these phrases, a model that considers only individual words[2] structure:

"You","may","say","$I'm$","a","$dreamer$","We","all","$live$","in","$yellow$","$submarine$"

And now we can count the frequency of each token:

- a = two times
- submarine = one time
- and so on.

In this example, the data is in the form of text that disregards the grammar and therefore word order. This approach of binding all the words together in the same "bag" therefore it takes the name of "Bag of Words" (BOW) [35].

Its aim is to classify documents, generate frequency count and group unique vocabulary for vectorization purposes. In this case, to do some sentiment analysis we need to label every token as "positive", "negative" or "neutral" and decide on the polarity of a sentence based on that. Using the BOW model we can say that a sentence has a positive meaning if there are more words with positive meaning than ones with negative meaning in it [35].

5.1.2 Term frequency-inverse document frequency

Inverse document frequency and term frequency (TF-IDF) takes the BOW as an assumption and goes a step further by choosing how important a word is in its sentence (and therefore in its document). TF-IDF does not take into account the order of words in a corpus [23]. It is made of two main parts:

$$TF(t) = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms}}$$

$$IDF(t) = log_{10}\frac{\text{Total number of documents}}{\text{Number of documents with term t in t}}$$

TF tells us how many times a term appears in a number of documents, therefore, its frequency. As you can see one drawback of TF is that, even if a token has little influence on the document sentiment such as "the", by giving equal weighting to every word it automatically imposes too much relevance for it [35].

IDF, on the other hand, shows the importance of that word in the document [35] by reducing the weighting of the most frequent terms while increasing that of rare ones.

[2]Also known as an uni-gram.

5.2 Considering context

John Firth[3] once said: "You shall know the word by the company it keeps".

As we have seen in the BOW model in section 5.1.1, some language models (LM) compute the probability of an individual token to appear in the whole text [35]. Unfortunately, this approach appears being too naive since context plays a major role in the construction of sentences.

The goal of the LM that we will analyze from now on is indeed to assign probabilistic sentiment to sentences by trying to catch context as well [35]. To do so, we will widely use the chain rule for conditional probability[4], whose formula is: P(A, B). This will allow us predict the chance of the token "eat" to appear in a document given the token "apple" [35].

Moreover, we are going to use the Markov Assumption [35], which states that we should calculate the probability of the next word by looking at the previous one. More complex versions use the Markov assumption and consider multiple words at the same time instead [35].

5.2.1 N-grams

Definition 5.2.1. N-gram is an approach that takes into account the combinations of N words with the aim of finding context.

How can we count the n-gram probability? By using the maximum likelihood estimate formula stated below:

$$P(w_i|w_{i-1}) = \frac{c(w_i, w_{i-1})}{c(w_{i-1})}$$

Consider the sentence:

"Imagine all the people living life in peace."

if we want its bi-grams (n=2), the output would be:

"Imagine all","all the", "the people", "people living", "living life", "life in", "in peace"

Why is the n-grams method great to analyze the context of a sentence? because it allows to compute the probability that a specific n-gram appears in a document. It is important to take into account that the more this tool is pushed, for example using a 6-gram, the more complex, time consuming and computationally expensive our model will be.

[3] famous linguist
[4] also called joint probability

5.3 Encoding words

Suppose to be looking at a restaurant menu, human beings can easily identify "Apple", "Chicken" and "Broccoli" inside of the text as foods, whereas "Tea", "Coffee" and "Vodka" as drinks. People can easily classify these items because we have a knowledge background that makes this possible. Machines lack this ability unless we teach them how to differentiate such items [35].

Encoding[5] means representing each piece of data into a format that an algorithm can use to make accurate predictions [9].

This technique is widely adopted in distributional semantics: a research area that tries to catch semantic similarities between words based on their distributional properties [35]. The **distributional hypothesis** tells us that words with similar distributions have similar meanings. Thanks to these tools, vectorization of words is possible.

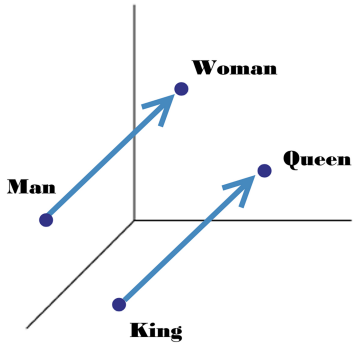

Figure 5.2: Word operations through vectors[6]

Thus, encoding is used for creating word embeddings, which are tools[7] used to vectorize words into feature vectors that can be fed into a NN [23]. As shown in Figure 5.2, word embeddings are learned representations of documents where similar words that have related meaning are represented by vectors close to each other.

Why is this relevant? because it means that we can perform mathematical operations on text in a way that actually make sense, such as in Figure 5.2. Namely:

$$w_{king} - w_{man} + w_{woman} \approx w_{queen}$$

This also helps us understand the clustering phenomenon[8] like in Figure 5.3, which by itself is fascinating.

[5] To encode literally means "convert to computer code" [9]
[6] https://www.oreilly.com/learning/capturing-semantic-meanings-using-deep-learning
[7] that we will closely analyze in the next sections
[8] which means: how do words group together?

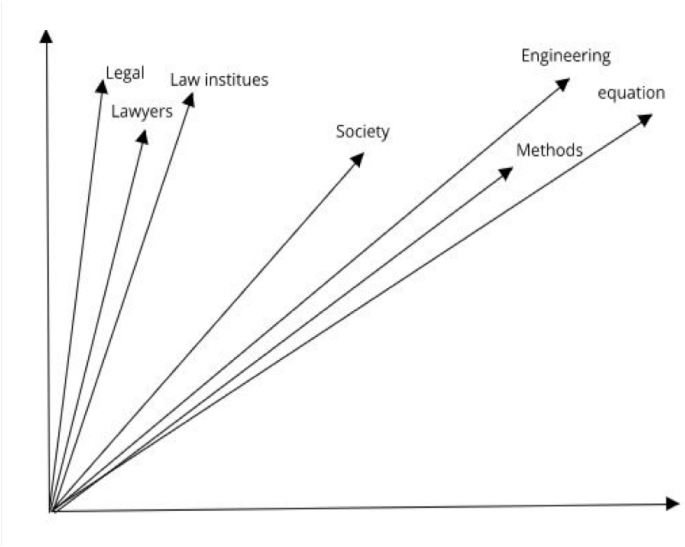

Figure 5.3: Text vectors clustering [35]

5.3.1 Label encoding

In a clothes shop, for example, a label is a small piece of plastic, paper or similar material attached to an t-shirt that provides information about it. Label encoding therefore, as the name suggests, encodes labels, which are specific categories in the data, which receive a number meant to represent them.

Choosing a category is a delicate process and the fact that those categories have natural ordered relationships can be an issue; this is due to the fact that computers are programmed to naturally give higher weights to categories with higher values. For instance:

Imagine having three types of foods we have seen before: "Apple", "Chicken" and "Broccoli". Label encoding assigns a number to each category: Apple = 1, Chicken = 2, and Broccoli = 3. Now suppose that the model needs to calculate the average among them: (1+3)/2 = 4/2 = 2. The results is that the average of Apple and Chicken is Broccoli [9].

This is a recipe for disaster!

5.3.2 One-hot encoding

One-hot encoding, by mapping categorical data attributes differently, helps us solve the previous problem by using a binary approach [9]: this encodes those labels into vectors

of length n. Visually:

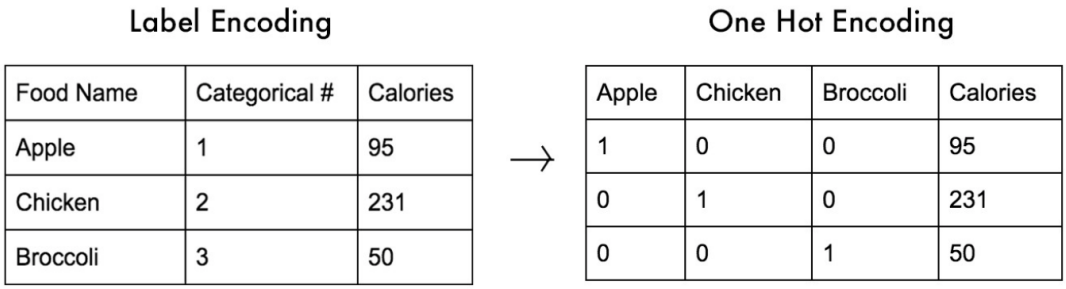

Figure 5.4: One-Hot Encoding [9]

What's the difference? Long story short, we have created an extra binary column for each category. The most part of machine learning models accept input data in the one-hot encoding format [35].

Suppose you apply one-hot encoding for the sentence "I love Eminem", this is not too difficult. However, what if you need to do the same thing on the whole vocabulary of Wikipedia? The result would be a huge matrix where operations would be extremely computationally expensive. We need another tool called Word embeddings, which are extremely efficient at implementing all of these operations [35].

5.4 Vector Distance

Sometimes, once the words have been vectorized, we may need to compute a scalar based on the two vectors taken into account which represents the distance between them. To do so, we have different options:

1. Dot product
2. Euclidean distance
3. Cosine distance
4. Trainable forms

Dot product

$$dist_{\text{dot}}(u, v) = u \cdot v = \sum_{i=1}^{d} u_{[i]} v_{[i]}$$

Euclidean distance

$$dist_{euclidean}(u,v) = \sqrt{\sum_{i=1}^{d}(u_{[i]} - v_{[i]})^2} = \sqrt{(u-v)\cdot(u-v)} = ||u-v||_2$$

Remark. In some cases the square root is omitted.

Cosine distance

$$dist_{cos}(u,v) = \frac{u \cdot v}{||u||_2||v||_2}$$

Trainable Forms

The issue with the methods presented above is that they represent fixed functions. Sometimes we need more flexibility. The benefit of using parametrized functions is that they can be trained to produce different distance values depending on the context [15]. A neural network, is the form of trainable distance that we need for our purposes [15].

5.5 Continuous Bag of Words

The continuous bag of words (CBOW) model is an evolution of the BOW that uses a basic form of feed-forward neural network. It can be thought as "predicting the word given its context" as shown in Figure 5.5.

Consider the sentence: "The cat climbed the tree". We could use the context words w_{n-2}, w_{n-1}, w_{n+1}, w_{n+2} for instance "the","cat","the","tree" as input for predicting "climbed" as the target word. The more the context words are considered, the input words we need to add to represent them properly [35].

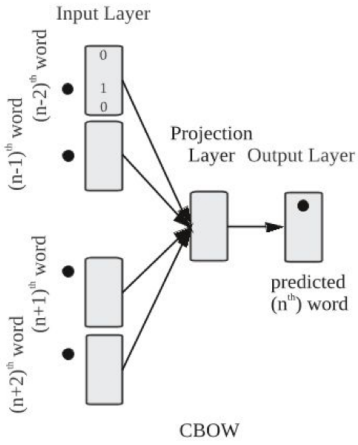

Figure 5.5: CBOW NN architecture: graphical representation [35]

The CBOW is a MLP where:

1. The error function of the CBOW is always the negative log likelihood of a word given a context:
$$p\left(w_O|w_I\right) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^{\top} v_{w_I}\right)}$$
where w_o is the output word while w_i is the context word.
2. The activation functions of the CBOW are linear.
3. There are multiple input layers and a single output layer.

5.6 Skip-gram

The skip-gram model reverses the perspective of the CBOW. In this case we try to "predicting the context given a word" as shown in Figure 5.6.

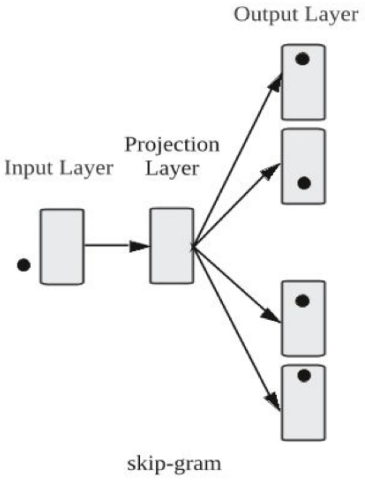

Figure 5.6: Skip-Gram NN architecture: graphical representation [35]

Take the same sentence in the CBOW example, in this case we will use "climbed" as input word to predict the "the", "cat", "the", "tree" context words. Apparently the Skip-gram model seems to be more accurate than the CBOW but at a greater computational cost since it takes more time to train such model [35].

5.7 Word2Vec

Word2Vec[9] is a famous model for word embeddings, which is built either by using the CBOW, the Skip-gram or a combination of the two and terminates with a Softmax Activation Function [35]. Formally:

Definition 5.7.1. Word2vec takes as its input a large corpus of text and produces a vector space through one-hot encoding, typically of several thousands of dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. At that point, the input-matrix acts as a "vocabulary index" that gets mapped to the text sequence that needs to be analyzed.

This model and its variations work through tensor manipulation (as shown in Figure 5.7) and heavily rely on human-tagged corpora [35].

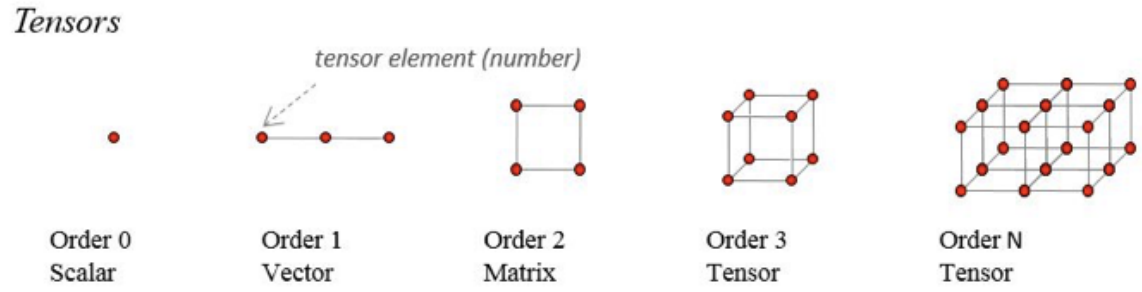

Figure 5.7: Graphical representation of a tensor [35]

This whole vectorization process comes with a great computational cost especially in terms of training time. A work around is the use of the so-called "pretrained word embeddings". In Layman's terms, they are big embedding matrices available online built by different researchers, for various purposes and often, with divergent assumptions.

Let's break the Word2Vec model down and try to understand more deeply the logic behind every piece:

1. Vocabulary Builder.
2. Context Builder.
3. Neural Network.

[9]This concept will be extensively described later, the purpose of this section is simply to give the intuition of what word vectorization methods like word2vec do.

5.7.1 Vocabulary Builder

The **Vocabulary Builder** takes raw-text data and find all the unique word tokens to create a vocabulary in the form of a one-hot encoded matrix. Word-removal procedures are useful when the user does not want to take into account the words that appear very rarely or stopwords [35].

5.7.2 Context Builder

The **context builder** takes as input the one-hot encoded matrix of the previous building block and outputs a context for the terms taken into account through the so-called context window.

This **context window** is a sliding window of various length: assume that you pick a six words window, then you have six words on the left side of the central word and the five words on its right side [35]. This process is called "word pairing" and a simplistic version is represented in Figure 5.8.

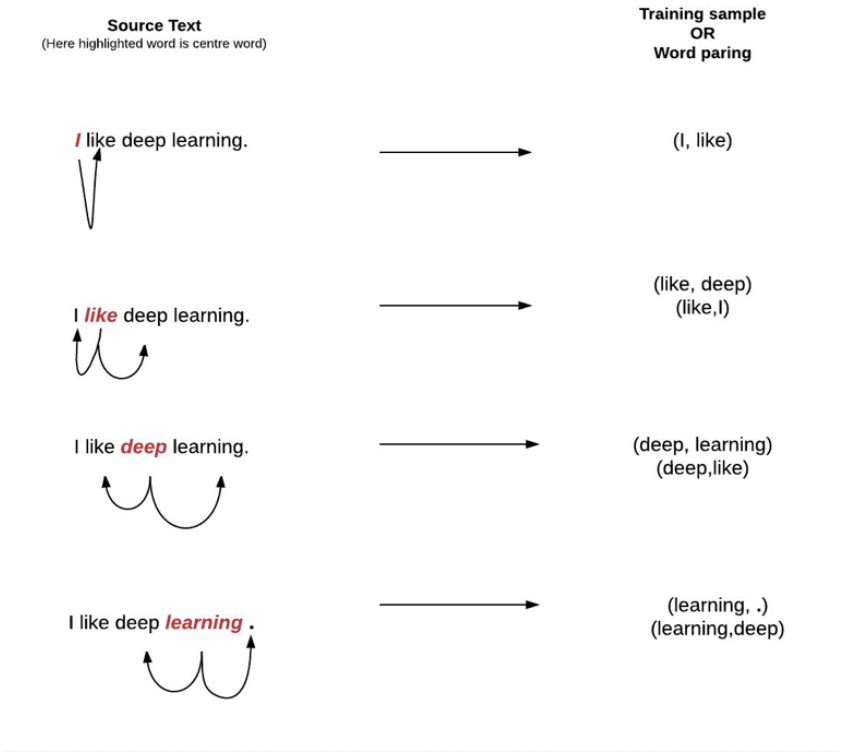

Figure 5.8: Word pairing: graphical representation [35]

Now we will go over some hyper-parameters[10] and tools that can be used to improve

[10] A hyperparameter is a parameter whose value is set before the learning process begins, in contrast with those parameters whose value is defined via the training process. Therefore it is arbitrarily chosen by the user.

the output's accuracy and efficiency of the context builder:

1. Dynamic window scaling.
2. Sub-sampling.
3. Pruning.

Dynamic window scaling

The intuition here is that the weight of the context words should not be uniform, since the closest words should hold more importance to predicting the target word than those that are far away. By providing this variable weight, you can fine-tune the context window to improve its performance [35].

Sub-sampling

Sub-sampling is another way to improve the quality of our training sample set. This method aims to remove randomly the most frequent and unnecessary words, namely "stop words" [35].

Pruning

Suppose you have a huge vocabulary to work with, to increase its accuracy you may want to get rid of those words who appear so rarely that can be considered useless. Pruning is used exactly for this purpose, to prune the training sample size and ultimately to improve its quality [35].

5.7.3 Word2Vec's Neural Network

The neural network structure of word2vec is inspired either from CBOW, Skip-Gram or a combination of the two [35]. These word-vector pairings built using the context window are the input into the neural network, which is going to learn the probabilistic relations among the data, given the frequency by which each word pairing appears. Its structure usually is [35]:

1. Input layer: the number of vectors matches the number of words available for training.
2. Hidden layer: its the dimensionality of the resulting word vectors.
3. Output layer: same number of neurons as the input layer.

To give a probabilistic meaning to our output the softmax activation function is used [35]. Word2Vec uses the sum of squared distance error as its loss function. Given the regression line:

$$y = mx + b$$

where m is the slope, b is where the line crosses the y-axis, x and y are respectively the independent and dependent variables. Therefore, once the line is drawn and we square[11] the sum of the distances between each initial point and the line, we obtain the "sum of squared distance error", which is:

$$\text{Error} = \sum_{i=1}^{N} (y_i - (mx_i + b))^2$$

How is it possible to minimize the error while we create the line of best fit? We use gradient descent! The problem with this model is that the one-hot encoding procedure in the Word2Vec input matrix is highly inefficient when dealing with huge datasets [35].

5.8 Global Vector

The Global Vector (GloVe) in an evolution of the Word2Vec algorithm where we realize that also ratios or word-to-word co-occurrence probabilities can encode meaning [18]. A natural and simple candidate, for instance, could be the "vector-difference" between 2 individual word vectors [18].

By doing so, just a single-pass through the entire corpus is necessary to collect the statistics we need. Even if shows better performance than Word2Vec, this model's process is still computationally expensive at the very beginning, especially for big corpora but then it gets more efficient later on [18]. What we might need is a leaner solution.

5.9 Pretrained and domain specific word-embeddings

The effectiveness of models such as Word2Vec and GloVe helped them to become the most popular tools used for language-related purposes. For this reason, it is easy to find many so-called "pretrained" word embedding models online.

The benefit of using them is that the user can load into its own model weights that have already been trained by people that (most likely) have access to stronger computational power. Their implementation is extremely simple and it usually translates in more accurate results [35]. Why because we are basically outsourcing the learning phase and this allows us to focus on more specific tasks.

However, there are also downsides to pretrained embeddings: Even though they are easy to use, some **domain-focused corpora** might show better performance when learning the word relations independently. This happens because pretrained word embeddings

[11] Those values are squared because we do not want to deal with negative values.

are usually built by their authors for specific reasons (such as translations or document classification) that may diverge from specific situations we need to solve. In these cases, the most common way to operate is to learn the word embeddings from zero by using an ad-hoc word-embedding [35].

How is this possible? By initializing each input word-vector randomly and letting the NN learn the relationships among words on its own.

5.10 Limitations of distributional methods

In the say we have explained everything so far we should have clarified that the definition of similarity, in distributional approaches, is operational: "words are similar if used in similar contexts" [35].

However, in the real world, there are many facets of similarity [35], for instance: consider the words cat, tiger and dog. By being both pets, cat is more similar to dog than to tiger, but at the same time it can also be considered closer to tiger given that they are both felines [35]. The distributional methods provide very little control over the kind of similarities they induce and that might change depending on the corpus that the model has been trained on [15].

Moreover, people are less likely to oversee information that are "obvious" for example: when people mention the word "sheep" in a conversation, they imply "white sheep" and do not explicitly mention the attribute "white" [15]. On the other hand, if the conversation moves to a sheep whose color is black we are more likely to hear "black sheep" mentioned often. The consequence is that a model trained on text data can be greatly misled by these behaviors that are commonly adopted [15].

These issues can be partially reduced by choosing carefully the training dataset and its preprocessing. In addition, distributional methods can suffer from corpus biases as well, that can be cultural or thematic [35]. The clearest example of this is polysemy. Take the word "tie" for instance, that can have different meanings depending on the context! We immediately notice that there is no such thing as a context-independent meaning [15]. This, as you can see, adds extra level of complexity that could be linked to a concept of quantum mechanics: quantum superposition[12].

So it is clear that each method has it's pros and cons: what makes all the difference is the user's ability to have full control over the dataset that he is working on and to adapt based on how its model is reacting [15].

[12]It is the property of being simultaneously in multiple states until it is measured.

Chapter 6

Model implementation

This model has been created to determine whether and to what extent future price movements can be predicted using sentiment analysis. By studying if alternative data, in the form of online news headlines, provide predictive clues that can be used to make better trading decisions.

A trading signal is, as previously introduced, a call for action to buy/sell an asset given a specific condition and after a certain analysis[1] that, in our case, is generated by a DL model.

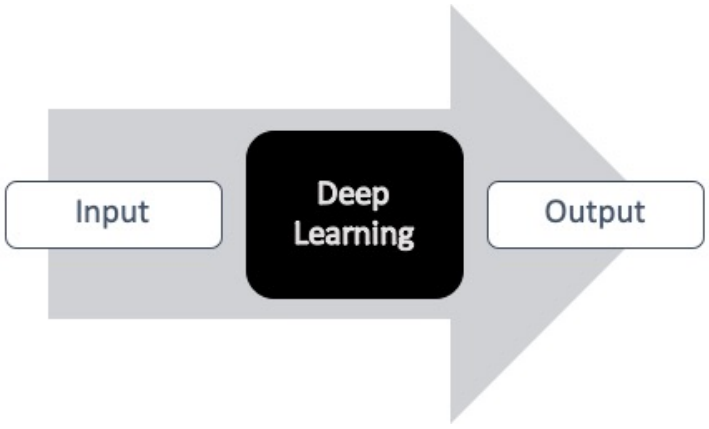

Figure 6.1: Deep Learning as a Black Box

When it comes to market behaviors, especially in the short term, we are trying to profit on the investors' "gut-feeling" rather than on other indicators. Since this phenomenon is something that we cannot always unequivocally define, we must reduce human judgment as much as possible by letting the algorithm learn directly, freely and dynamically from the daily movements of a security. Deep learning is well known for being considered a black-box, as shown in Figure 6.1, which means that it is a system, whose

[1] https://www.investopedia.com/terms/t/trade-signal.asp

inputs and outputs are known, but its internal behaviors are not completely understood[2].

Our job is to open and have a better understanding on how this mechanism behaves for our purposes. Before moving on with the detail of the model we need to clarify some of the tools and assumption that have been used.

6.1 Instruments used

The programming language that has been used for this project is Python 3, an open general-purpose programming language that has seen wide adoption among firms, institutions and researchers in the last few decades[3].

NLTK

The Natural Language Toolkit (NLTK) is a python library used to work with natural language data. It consists of corpora, lexical resources and with a suite of text processing libraries for classification, parsing, and so on. Thus, it quickly became one of the most popular APIs for linguists, researchers, and hobbyists[4].

Numpy

NumPy is an open-source and high-speed Python library for scientific computing which makes wide use of linear algebra, N-dimensional arrays, etc. It can also be used for multi-dimensional generic data, since data-types can be defined by the user[5].

Pandas

The Python Data Analysis (Pandas) is an open-source python library that provides easy-to-use data structures and data analysis tools built on top of NumPy[6].

Keras

Keras is a high-level deep learning library, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Since being able to go from idea to result quickly is crucial for research tasks, it was developed to pursue a smoother prototyping process[7].

[2]http://www.businessdictionary.com/definition/black-box.html
[3]https://www.python.org/
[4]http://www.nltk.org/
[5]http://www.numpy.org/
[6]https://pandas.pydata.org/
[7]https://keras.io/

6.2 The Implemented System

As shown in Figure 6.2, we will start by seeing the Dataset that has been chosen. Right after, we will analyze which preprocessing operations have been implemented to ease the computational effort for the model. Then, we will see the all the components of the Deep Learning model that has been put in place. In the end, the results will be presented together with a real-case scenario for the model.

Figure 6.2: The Implemented System

As stated in the previous chapters, the performance of a deep learning model improves the more data it can train on. The main barrier which can be encountered is, indeed, physical given the fact that DL algorithms depend on the amount of computing power that can rely on [35]. For this reason, the algorithmic trader that wishes to use this technology, needs to find the correct balance between his tools' capability and the accuracy he desires. To sum up, when working with neural networks, we encounter some limitations that might affect our results [35]:

1. Dimension of the dataset: the more (meaningful) examples the model is able to be trained on, the better it will perform.

2. Computing power availability: A NN could have a big amount of dataset in which he could train on, but it also needs the computing power necessary to be able to complete its training in a reasonable time.

3. Model complexity: The model should be complex enough to avoid a high level of bias, but not too much so that it can avoid excessive variance.

In our case, the tests have been done in a Mac-Book Pro 2016 with 2,9 GHz Intel Core i5. In addition, note that the complete version of the code is available on this GitHub folder: https://github.com/Mattia9494/Sentiment-analysis-with-AI.

6.3 The Dataset choice

The choice of a correct dataset to work with is one of the most important decisions when dealing with any NLP issue and it has been indeed one of the biggest challenges for super-

vised DL models. At the time of writing, there is a discrete amount of free human-tagged databases available online. The recent introduction of the "Dataset Search" website[8] by Google, a web search engine focused on finding databases, simplified this process.

The majority of the publicly available datasets relates to products or movie reviews such as, for instance, the popular "IMDB Movie review Dataset" or "Amazon Reviews". Despite their high quality and wide adoption among the deep learning community, they are not directly linked to stock markets.

When facing the final database choice there were two main candidates:

1. The Sentiment140 Database (https://www.kaggle.com/kazanova/sentiment140) from Kaggle, containing more than one million generic tweets with either positive, neutral or negative sentiment.

2. The Dow-Jones Industrial Average (DJIA)[9] Database from Kaggle[10] which contains 25 news headlines each day with either a positive or negative sentiment based on whether the DJIA increased or decreased. The news have been scraped from the 25 most upvoted ones by the community on Reddit WorldNews by the author.

The DJIA database has been chosen for several reasons. Despite the strength of the Sentiment140 dataset is certainly its large data quantity, it showed some weaknesses both at linguistic and financial level that the DJIA did not have and that cannot be overlooked.

Let's see the composition of the DJIA dataset:

1. A first column of dates from 08-08-2008 to 01-07-2016 for a total of 1989 lines[11].

2. A second column of human-tagged labels, in which:

   (a) One: is expressed when the DJIA increases or stays the same on the same day.

   (b) Zero: in case of a daily decrease of the DJIA.

3. The next 25 columns represent the most relevant news for that day in the form of strings. During the preprocessing phase these news strings will be merged together in a unique string for each row. By doing so we will obtain the general market signal that we will try to predict.

[8] https://toolbox.google.com/datasetsearch

[9] The Dow Jones Industrial Average (DJIA) index, invented in 1896, is a price-weighted average of the thirty most important companies traded on the New York Stock Exchange and the Nasdaq.

[10] https://www.kaggle.com/aaron7sun/stocknews

[11] Note that these numbers are relative T0 case, for all the others, depending on the one considered there might be changes for a few days.

In the model built on the DJIA dataset we assume that the news analyzed report the social events that happen in the world in that specific date. Moreover, we assume that these news providers are:

1. Not politically biased, therefore that the journalists report the news objectively without distorting facts for benefits related to personal glory or their journal's interests.

2. Journalists do not passively copy the work from their colleagues, which translates in the absence of any "herd effect" in the use of specific words or expressions.

This analysis takes into account the opening and adjusted closing prices in the considered days. Despite being true that certain type of news can have an intraday price effect that can disappear later on, working with intraday data points may cause an excessive complexity that goes beyond the scope of this study.

With this work, we are not simply interested in knowing whether a sentence is positive or negative from a linguistic stand point, but our attention is focused on the price-change effect that that news may cause. For instance, a news that states: "Today's missilistic attack has severely decreased the medical supply for the population of country XXX". Linguistically, it might sound like having negative meaning due to the word "decreased". However, the effect on the DJIA could be positive, if the investors start buying certain stocks such as 'UNH'[12] that are part of the DJIA. In order to eliminate any risk and reducing human assumptions as much as possible, we will let the DL algorithm decide the sentiment for such specific cases.

Why did we not trade on tweets? One can argue that a tons of tweets can contain relevant information in the form of underlying market buzz, however this connection is not immediate. While a tweet from president Trump can be impactful on an investor's life, it is probably not the case for your neighbor's opinion discussed online.

One could argue that by studying social media "influencers" [3] it is possible to achieve financially relevant results; nevertheless this is particularly true for news providers such as newspapers that, backed by a history of providing this type of service, have more authoritativeness.

Moreover, if on the one hand a news headline is characterized by the use of simple terms (and sometimes exaggerations) in order to attract the reader's attention; On the other hand there is an intrinsic controversy when dealing with tweets [37]. The limit of 280 characters[13] and the use of instruments such as hashtags have deeply influenced the way that users express meaning on the Twitter platform [37].

[12]United Health Group Inc

[13]It was limited at 140 characters in the past.

Finally, hashtags are great at expressing ironic content [21] in the form of paralanguage. They would, for instance, be placed specifically to reverse the sentence's initial meaning, for example: "I love president Trump! #TrumpImpeachmentParty". These behaviors not only show how linguistically complex dealing with these situations can be [37], but also how conceptually wrong the direct hashtag removal procedure is, at least in Twitter conversations. One of the most challenging tasks for NLP models is, in fact, identifying irony and jokes. This is due to the fact that this type of algorithms does not only need to understand the denotation of the language, but also the hidden context that surrounds it [35].

6.4 The studied model

This analysis focuses on aggregate market indicators, rather than individual securities [3]; in particular we will try to predict the movements of the DJIA based on the 25 most popular daily news that have been published between 08-08-2008 and 01-07-2016. In our model we will compare Two cases:

6.4.1 Case A

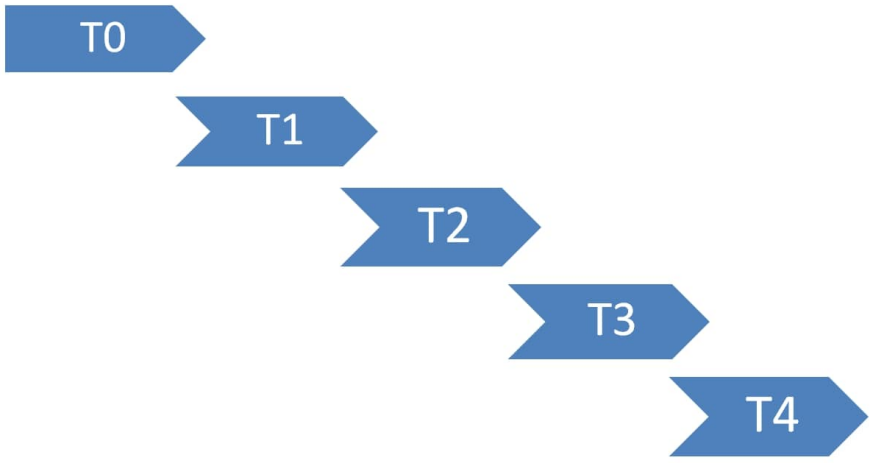

Figure 6.3: Case A

As shown visually in Figure 6.3:

1. T0 case: If, based on the news published today, today's Adjusted closing price[14] is higher (daily increase) or lower (daily decrease) than today's opening price.

[14]Note that we used the adjusted closing price because, on average, a stock's price is affected by some corporate actions such as dividends, stock splits and so on.

2. T1 case: If, based on the news published today, tomorrow's Adjusted closing price is higher (daily increase) or lower (daily decrease) than tomorrow's opening price.
3. T2 case: If, based on the news published today, the Adjusted closing price in 2 days is higher (daily increase) or lower (daily decrease) than the opening price in 2 days.
4. T3 case: If, based on the news published today, the Adjusted closing price in 3 days is higher (daily increase) or lower (daily decrease) than the opening price in 3 days.
5. T4 case: If, based on the news published today, the Adjusted closing price in 4 days is higher (daily increase) or lower (daily decrease) than the opening price in 4 days.

6.4.2 Case B

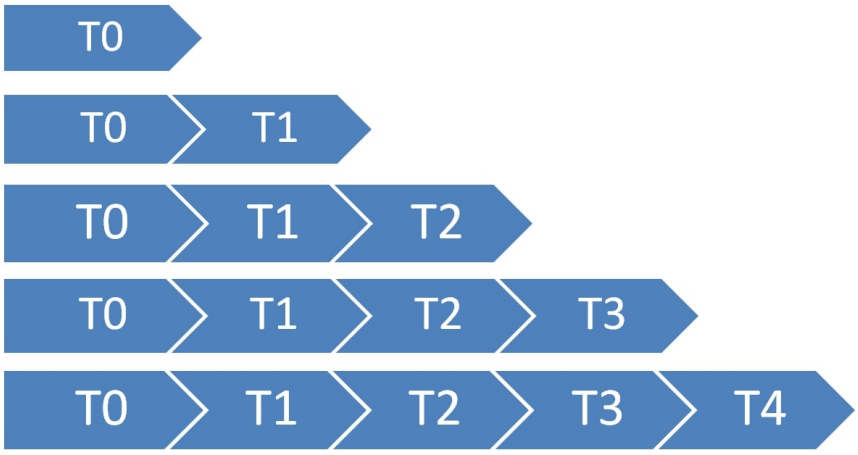

Figure 6.4: Case B

As shown visually in Figure 6.4:

1. T0 case: Same case as in Case A.
2. T1 case: If, based on the news published today, tomorrow's Adjusted closing price is higher or lower than today's opening price.
3. T2 case: If, based on the news published today, the Adjusted closing price in 2 days is higher or lower than today's opening price.
4. T3 case: If, based on the news published today, the Adjusted closing price in 3 days is higher or lower than today's opening price.
5. T4 case: If, based on the news published today, the Adjusted closing price in 4 days is higher or lower than today's opening price.

6.5 Preprocessing

The columns representing the most relevant news headlines for each day are merged together in a single string that represents the unique market signal. In this scenario the preprocessing phase is crucial to reduce the number of independent variables, namely the word tokens, that the algorithms need to learn. From the point of view of preprocessing these were the choices that have been made:

1. Remove the stopwords enumerated in the Section 2.3.1.
2. Transform every element in lower cases because the model is cases sensitive, which means that could see "dog" and "Dog" as two different entities.
3. Remove punctuation.
4. Remove words that rarely appear in the text, by keeping only the 3000 most frequent words appearing in the vocabulary.

To do so many procedures are put in place such as 'word-removal', which consists in getting rid of those words that are considered less important, in our case stop words and rare ones. This has been mostly achieved thanks to the use of Regular expressions and NLTK tools.

Once all elements have been removed, we obtain a totality of 30625 unique words and of those, only the 3000 most frequent ones were kept. This number has been decided because, we needed an amount words that a non-native English speaker needs to understand basic concepts[15].

6.6 Deep Learning Architecture

A NLP neural network cannot directly read strings of text, it needs to receive a two-dimensional matrix of numbers to do so. Thus, before being parsed, we need to transform this overall string into a sequence of individual tokens for every row, that the model will see as numbers. To do so we use the Keras tokenizer, whose function is to vectorize a text corpus, by turning each text into a sequence of integers.

At this point we encounter a problem: it would be impossible to feed the model with sequences of tokens in the form we currently have given that each row has a different length. The workaround is through the so called "sequence padding". This is how it works: first we need to arbitrarily choose a value that represents the maximum sequence length[16] allowed that in our case is 110; this number has been chosen by trial and error.

[15] https://www.ef.com/wwen/english-resources/english-vocabulary/top-3000-words/
[16] This number has been decided by trial-and-error.

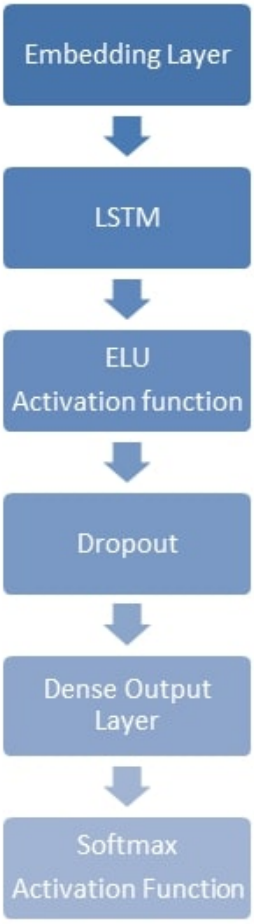

Figure 6.5: Deep Learning Architecture

After that, all those rows that contain sequences smaller than that number, will be filled with zeros at the beginning of the sequences until 110 is reached. For instance, if a row has only 23 tokens, then 77 zeros will be added at the beginning of the sequence in order to reach the maximum length of 110.

At the end of this process we obtain a 2D-matrix of dimension (1989,110) in which 1989 is the number of rows of our dataset and 110 is the maximum length that with which we constrained the algorithm.

What about the labels? That column of ones and zeros representing the increase or decrease of the DJIA, in the meantime, has been transformed into a categorical matrix. This means that thanks to the "to_categorical" function in Keras converts this into a matrix with as many columns as there are classes, in our case two. The matrix that comes out of it is of shape (1989,2), where the first number is the number of rows and the second number is the number of categories we need to predict.

One of the biggest benefits of using this dataset is its balance between labels. In fact, we have nearly 50% of labels for each category in our datasets, which means that the neural network will be able to train on roughly the same amount of examples for each class.

Our neural network is composed of 3 layers:

1. An embedding layer.
2. A hidden layers composed by LSTM neurons.
3. An output layer.

6.6.1 Embedding Layer

Let's start with the Embedding layer, which is the input of the model, whose job is to receive the two-dimensional matrix that we have been talk about and spit out a three-dimensional one to be fed in the next layer.

How does this happen? First of all, the embedding layer receives as input the number of words in the training set and the size of each sequence of words for each row. At this point the user needs to arbitrarily decide an output dimension that will represent the "depth" of the 3D matrix. Once this is decided, the embedding layer will initialize weights based on the output dimension stated.

How are these weights initialized? there are two main approaches:

1. Loading a pretrained word embedding such as Word2Vec.
2. Initialize the weights randomly.

There is not a one fits all "correct" way to initialize weights. The benefit of using pretrained embeddings is that they are created by academics or institutions with much more computational power that you could hope for. The drawback is that these pretrained embeddings are usually made for specific purposes that might differ from what you are trying to achieve. If a pretrained word embedding has been created to solve a translation issue, for instance, it might not be useful for us, despite its completeness and high quality.

In our case, both paths have been experimented. Firstly the weights have been initialized thanks to a GloVe embeddings[17], uploaded from those published by the University of Stanford[18]. Unfortunately, this did not lead to any relevant improvement in the performance of our neural network and I opted for a random initialization that follows a

[17]Note that the experiments attempted used the GloVe method built on all the Wikipedia word universe. Different or better results can be in theory found with more thematic specific word embeddings, which at the moment of this writing, I have not found.

[18]https://nlp.stanford.edu/projects/glove/

Uniform Distribution instead[19].

The reason could have been justified by the fact that the pretrained weights have been generated by using a generalist selection of words, such as "all the vocabulary available on Wikipedia", that might not perform well in the domain specific problem that we are facing. All of these notions may sound complex at first, therefore I realized that an example might be much more straightforward to have a better understanding on how the words have been vectorized. let's take two sentences:

1. I want to eat pasta.
2. We want to eat again.

That would form a vocabulary where each wor is linked to an index:

Index	Relative word
0	"I"
1	"want"
2	"to"
3	"eat"
4	"pasta"
5	"we"
6	"again"

Once they will be encoded as single integers they will look like this:

1. $[0, 1, 2, 3, 4]$
2. $[5, 1, 2, 3, 6]$

Now let's assume that we have an embedding layer that accepts the whole vocabulary of seven words, accepts a sequence length of five, like the number of words in each sentence and requires an embeddings dimension of two random weights. The embedding layer can be, thus, seen as a table used to map integers to embedding vectors:

Index	Embedding
0	$[2.1, 4.0]$
1	$[1.0, 5.1]$
2	$[1.9, 4.0]$
3	$[1.2, 3.0]$
4	$[3.1, 2.3]$
5	$[1.6, 2.6]$
6	$[5.0, 2.9]$

[19]Which is implemented as standard in the Keras Embedding Layer that can be found in https://keras.io/layers/embeddings/

Therefore according to this embedding layer, sentence n°2 is written as:

$$[[1.6, 2.6], [1.0, 5.1], [1.9, 4.0], [1.2, 3.0], [5.0, 2.9]]$$

Coming back to our case, the embedding layer in the T0 case will receive as input a (3000, 110, 256) where 256 is the output dimension that has been chosen for the embedding layer.

6.6.2 Hidden Layer

After all this computation is done, the matrix is pushed to the LSTM hidden layer that is composed by 256 neurons and works exactly like it has been explained step by step in Section 4.5.5. The weights within the LSTM hidden layer are randomly initialized following the Glorot Uniform Initialization procedure seen in Section 4.7.1, while the biases have been initialized as zeros[20]. Outside the LSTM hidden layer, the ELU activation function that we have seen in Section 4.4.5 proved to be the most effective activation function in our case.

Moreover, a Dropout, as shown in Section 4.6, has been implemented as regularization method both outside and within the LSTM cells[21] given its simplicity and effectiveness. This procedure has been particularly useful to promote generalization of the model and reduce overfitting.

6.6.3 Output Layer

At the end of all these steps, the array will be passed to the output layer, made of two dense neurons. Finally, the softmax activation function, shown in Section 4.4.6, following this layer will add a probabilistic meaning to the data and state a prediction.

Once the model's structure has been determined, it needs to be properly compiled by choosing the optimizer used for the backpropagation and gradient descent procedures. In our case, it is ADAM that we have met in Section 4.3.7.

How is the model's performance evaluated? By using two important metrics:

1. The cross-entropy loss function: that we have previously encountered in Section 3.3.1 and that represents a summation of the errors made for each example received in the training or validation set. Therefore, the lower, the better.

2. The accuracy: this metric, as described in Section 3.3.2, is determined when the learning phase is complete.

[20]This inizialization choices are a standard within the Keras LSTM layer available in: https://keras.io/layers/recurrent/

[21]In this case it takes the name of "recurrent dropout"

At this point in time, we assembled a generalist model which is complete, compiled and we defined its evaluation metrics used for its evaluation need to fit our database into it to start the training process. To do so, it is necessary to split the initialized weight-matrix between training and validation set. The training set will be the matrix that we will use to train the network while the validation set will be used to see how good it performs.

6.7 Fitting the data in the Deep Learning model

In order to divide training and testing set a 80/20 split has been implemented, which means that the network will be trained on 80% of the dataset while it will be validated on the last 20% of the rows.

At this point we have almost everything we need: split dataset, model structure, evaluation metrics, optimizer and so on. Nevertheless, we still need to clarify a few crucial missing pieces:

1. Number of Epochs.
2. Batch Size.
3. Number of Iterations.
4. Shuffling the training set.

We need elements such as those explained above only when the data is too hard for the machine to chew, which is a common issue in DL [31]. In other words, we need to do that when we cannot pass all the information to the computer at once without putting your hardware at risk.

Therefore, to overcome this problem we use the "Divide et Impera" approach by dividing the data into small pieces and give it to our computer one at the time. This will help the model ease the computational effort of updating the weights that change at every step to fit into the whole database [31].

Number of Epochs

The number of epochs represents how many times the whole dataset is passed forward and then backward through the neural network as explained in Section 4.3.1 with Back-propagation.

In our T0 model, the best performance has been achieved after three epochs. At first glance it does not seem intuitive to think that passing the entire dataset through a NN may not be enough [31]. However, when using a limited dataset and to optimize the training process we are using Gradient Descent which is an iterative process. Thus,

updating the weights with a single epoch is usually not enough to train the model effectively [31].

Batch Size

Exactly like this thesis needs to be divided into chapters and paragraphs to be easily understandable, you need to divide a database into a certain number of parts named Batches [31].

Batch Size is therefore the dimension of each of these parts: in T0 the Batch Size that showed the best performance is 64. This means that the training set made of 1611 rows has been splitted in parts that were 64 rows big and used as input into the model one by one.

Number of Iterations

Finally, the number of iterations is a consequence of the Batch Size since it represents the number of batches necessary to complete one epoch [31]. Therefore, our T0 model shows $1611/64 = 26$ iterations to complete one epoch.

Note that the batches that will be fed into the model will have a size of 64 until the very last batch will have a size of the remaining 11 rows.

How many epochs, iteration do we need? How big should the batch size be? And how many neurons should the hidden layer have? Unfortunately, when it comes to deep learning problems, there is no right answer.

Remember, once again, that different datasets require different approaches and, in most cases, the best one can only be discovered through a trial and error procedure. Nevertheless, the numbers of epochs required is related to how diverse, big or complex the database is [31].

Shuffling the Training Set

Another useful instrument is the random shuffling of the training set, which is used in our case and particularly when we want to ignore the temporality among the dataset elements: this means that if we saw a daily increase yesterday, we assume that it has no influence on today's price movement [35].

Intuitively, exactly like when shuffling a deck of cards to play a different hand, the model will shuffle the rows of the training set to work more generally after each Epoch. This procedure allows the model to reduce overfitting not to being constrained by the row order of the data during the learning phase [31].

6.7.1 Callbacks

Before starting the training, we could implement some particularly useful tools that are available in the Keras API. One of the most useful, when it comes to selecting the best DL models, are the callbacks. A callback is a set of functions to be applied at given stages of the training procedure.

They are used to look inside the internal states of the model during the training[22] and impose some actions or restrictions. The ones that have been implemented in our case are:

- Checkpointer, which saves the model at each epoch, which is really useful to see at which iteration the model performs at its best.
- Early stopping's job is to freeze the model when, after a certain number of epochs, is not improving anymore. This is particularly useful, because it allows you to set a high number of epochs and let the model run by itself, since you know that it will stop automatically if necessary.

6.8 Seeding the model

As you might have guessed, neural networks are stochastic [5]; they are unstable by design since they make a wide use of randomness, which is deeply embedded in DL algorithms performs better with than without [5]. In fact, we see randomness in weight initialization, in regularization, in optimization and so on [5].

Then how is it possible to obtain reproducible results? After all this struggle in preprocessing, model structuring and regularization procedures, the user not only needs to be able to see which tools are concretely useful to improve its model's performance, but he also needs to be able to make real world predictions [5]. The traditional solutions to this issue are two: repeat the experiment multiple times or seed the random number generator. For our case, the second way has been pursued.

Random numbers are generated using a pseudo-random number generator inside your computer [5]; a random number generator is therefore a function that will generate a huge sequence of numbers that are random enough for general purposes. These generators need a seed to kick off the process, and it is common to use the current time in milliseconds as default since it ensures that different sequences of random numbers are generated each time the code is run, by default [5].

This seed can also be specified with a defined number to ensure that the same sequence of random numbers is generated each time the code is run. To do so, both the NumPy

[22] https://keras.io/callbacks/

and TensorFlow random generators have been seeded and this is exactly the method that has been implemented to obtain a deterministic model instead of a stochastic one [5].

6.9 Hyperparameters tuning

One last thing, before showing the final results, the Hyperparameters tuning method that has been applied was "Babysitting", as explained in Section 4.9.1, and each version of the model had its own.

Specifically, the ranges where each hyperparameter has been tuned are:

1. Dimension of the embedding input layer: [32, 64, 128, 256, 512]
2. Number of neurons in the LSTM hidden layer: [32, 64, 128, 256, 512]
3. Dropout: [0, 0.1, 0.2, 0.3, 0.4, 0.5]
4. Recurrent dropout: [0, 0.1, 0.2, 0.3, 0.4, 0.5]
5. Batch size: [32, 64, 128, 256, 512]
6. Number of epochs: [1, 2, 3, 4, 5]

Case A - Hyperparameters

Version	T0	T1	T2	T3	T4
Embedding Dimension	256	128	64	128	512
Hidden Layer dimension	256	64	256	256	256
Dropout	0.3	0.3	0.3	0.3	0.5
Recurrent dropout	0.3	0.1	0.2	0.3	0.3
Batch Size	64	64	64	32	64
Epochs	2	2	2	4	3

Case B - Hyperparameters

Version	T0	T1	T2	T3	T4
Embedding Dimension	256	32	64	512	512
Hidden Layer dimension	256	128	64	32	32
Dropout	0.3	0	0.3	0.4	0.4
Recurrent dropout	0.3	0.1	0	0.2	0.5
Batch Size	64	64	64	64	128
Epochs	2	2	3	3	3

6.10 Model Results

Case A - Accuracy

After different experimentation the best results for:

1. 57,94% in the validation set for the T0 case.
2. 55,85% in the validation set for the T1 case.
3. 54,79% in the validation set for the T2 case.
4. 53,46% in the validation set for the T3 case.
5. 53.60% in the validation set for the T4 case.

Case B - Accuracy

After different experimentation the best results for:

1. 57,94% in the validation set for the T0 interval.
2. 55,17% in the validation set for the T1 interval.
3. 55,59% in the validation set for the T2 interval.
4. 55,47% in the validation set for the T3 interval.
5. 57.75% in the validation set for the T4 interval.

6.11 Practical application

How would this model behave in a practical application? in particular we are interested in whether it would be able to correctly predict the DJIA based on specific news related to major political events that might have affected the globe. Note that every event chosen after the date 01-07-2016 which is the last day analyzed in the model's database; this guarantees the training data, testing data and application cases do not overlap in any case.

The political events chosen are closely related to international election in western European countries, specifically events that would represent a profound change in society that would influence either Europe or the USA. For this reason, the events chosen for the political election case were:

- The American election with the rise of the Trump in power.
- The French election with the rise of the Macron in power.

- The Italian election with the rise of populist parties M5S and Lega.
- The German election with Angela Merkel being confirmed to power.
- The result of the Midterms in the USA, with the Democrats taking control of the House of representatives.

While, in the second analysis, we decided to look more closely into Trump's presidency and see the reaction of the different political actions that have been pursued and that caused great attention in the world and within the country. For this reason, the events chosen for the "Trump's policies" case were:

- The Summit in Singapore between Trump and Kim Jong-Un.
- Trump signs for tariffs on steal and aluminum.
- Trump formally shows the intention of US withdrawal from the Paris Agreements on climate change.
- Trumps signs a big tax cut that was particularly beneficial to big corporations.
- Trump starts the government shutdown to make pressures for building the wall.

Disclaimer: Due to the lack of relevant worldwide events such as those enumerated in our practical implementation, I opted to present a number of limited, but highly focused cases rather than using larger samples but less linked with one another.

In addition, some specific events may appear in days where for example "T2 is a holiday", in such cases I could have proceeded in three ways:

1. Eliminate that date completely, signaling it with an "X" symbol.
2. Choose another relevant event.
3. Use linear interpolation between the preceding an following opening/adj.close price.

Given the difficulty to find worldwide relevant events that might have an impact on the US economy, I opted for the third option.

6.11.1 Political Election - Case A

Prediction

Now we will see what the model predicts for every political election event:

Version	T0	T1	T2	T3	T4
American Election	Bearish	Bullish	Bullish	Bullish	Bearish
French Election	Bullish	Bearish	Bullish	Bullish	Bearish
Italian Election	Bullish	Bullish	Bullish	Bearish	Bullish
German Election	Bullish	Bullish	Bullish	Bullish	Bullish
Midterm Election	Bearish	Bullish	Bullish	Bullish	Bullish

DJIA behavior

Now we will see how the DJIA really behaved in that same day:

Version	T0	T1	T2	T3	T4
American Election	Decreased	Increased	Increased	Increased	Increased
French Election	Increased	Decreased	Decreased	Decreased	Increased
Italian Election	Increased	Decreased	Increased	Increased	Increased
German Election	Decreased	Decreased	Decreased	Increased	Increased
Midterm Election	Increased	Increased	Decreased	Decreased	Decreased

Results

By comparing each prediction with how the DJIA actually behaved:

- T0: 3/5 correct cases.
- T1: 3/5 correct cases.
- T2: 2/5 correct cases.
- T3: 2/5 correct cases.
- T4: 2/5 correct cases.

6.11.2 Political Election - Case B

Prediction

Now we will see what the model predicts for every political event:

Version	T0	T1	T2	T3	T4
American Election	Bearish	Bullish	Bullish	Bullish	Bullish
French Election	Bullish	Bullish	Bullish	Bullish	Bullish
Italian Election	Bullish	Bullish	Bullish	Bullish	Bullish
German Election	Bullish	Bullish	Bearish	Bearish	Bullish
Midterm Election	Bearish	Bullish	Bullish	Bearish	Bullish

DJIA behavior

Now we will see how the DJIA really behaved:

Version	T0	T1	T2	T3	T4
American Election	Decreased	Increased	Increased	Increased	Increased
French Election	Increased	Decreased	Decreased	Decreased	Decreased
Italian Election	Increased	Increased	Increased	Increased	Increased
German Election	Decreased	Decreased	Decreased	Decreased	Increased
Midterm Election	Increased	Increased	Increased	Decreased	Decreased

Results

By comparing each prediction with how the DJIA actually behaved:

- T0: 3/5 correct cases.
- T1: 3/5 correct cases.
- T2: 4/5 correct cases.
- T3: 4/5 correct cases.
- T4: 3/5 correct cases.

6.11.3 Policies during Trump's presidency - Case A

Prediction

Now we will see what the model predicts for every political event:

Version	T0	T1	T2	T3	T4
Deal with North Korea	Bearish	Bullish	Bullish	Bullish	Bearish
New Tariffs	Bullish	Bullish	Bearish	Bullish	Bearish
Paris Agreement	Bearish	Bullish	Bullish	Bullish	Bullish
Tax cut	Bullish	Bullish	Bullish	Bullish	Bearish
Shutdown	Bullish	Bullish	Bullish	Bearish	Bearish

DJIA behavior

Now we will see how the DJIA really behaved:

Version	T0	T1	T2	T3	T4
Deal with North Korea	Decreased	Decreased	Decreased	Decreased	Decreased
New Tariffs	Increased	Increased	Increased	Increased	Decreased
Paris Agreement	Increased	Increased	Increased	Increased	Decreased
Tax cut	Decreased	Decreased	Decreased	Decreased	Increased
Shutdown	Increased	Increased	Increased	Decreased	Increased

Results

By comparing each prediction with how the DJIA actually behaved:

1. T0: 3/5 correct cases.
2. T1: 3/5 correct cases.
3. T2: 2/5 correct cases.

4. T3: 3/5 correct cases.
5. T4: 2/5 correct cases.

6.11.4 Policies during Trump's presidency - Case B

Prediction

Now we will see what the model predicts for every political event:

Version	T0	T1	T2	T3	T4
Deal with North Korea	Bearish	Bullish	Bullish	Bearish	Bullish
New Tariffs	Bullish	Bullish	Bearish	Bullish	Bullish
Paris Agreement	Bearish	Bullish	Bullish	Bearish	Bullish
Tax cut	Bullish	Bullish	Bullish	Bullish	Bullish
Shutdown	Bullish	Bullish	Bullish	Bullish	Bullish

DJIA behavior

Now we will see how the DJIA really behaved:

Version	T0	T1	T2	T3	T4
Deal with North Korea	Decreased	Decreased	Decreased	Decreased	Decreased
New Tariffs	Increased	Increased	Increased	Increased	Increased
Paris Agreement	Increased	Increased	Increased	Increased	Increased
Tax cut	Decreased	Decreased	Decreased	Decreased	Decreased
Shutdown	Increased	Increased	Increased	Increased	Increased

Results

By comparing each prediction with how the DJIA actually behaved:

1. T0: 3/5 correct cases.
2. T1: 3/5 correct cases.
3. T2: 2/5 correct cases.
4. T3: 3/5 correct cases.
5. T4: 3/5 correct cases.

6.12 Comments on the results

After this practical implementation we need to make some considerations under various aspects:

First of all we notice that the level of accuracy of this model is really low, both in Case A and B our peaks stick around 58%, which is basically a flip of a coin. Moreover, in both version of the model the highest accuracy appears in the T0 case; behavior that suggest that, even slightly, shorter periods of time should be preferred for forecasting.

Moreover, we notice a tendency, that the accuracy in case A, in general, tends to gradually decrease the further away in time we go, since it gradually passes from 57% to 53%. On the other hand, case B shows a less evident decrease that ends with an unexpected increase in T4. For this reason it might be interesting to continue, with further research, studying how such model behaves over longer periods of time.

Coming at the analysis of the practical application we immediately notice that, exactly as expected, given the accuracy level that have been encountered, there is not a single case where the algorithm achieved a 100% forecasting level. On the one hand, looking at the "Political Election" Case we immediately notice three things:

1. Both cases showed good performance, but not exceptionally high.
2. Case A showed a decrease in line with the behavior shown in the accuracy level.
3. Case B, in general appeared to perform better than Case A, as expected.

On the other hand, in the "Trump's Actions" case we notice:

- Both cases performed in the similar way that we have seen in the previous case.
- Case B, in general appeared to under-perform compared to what we have encountered in the Political Election Case.

An alternative to this second analysis could be a study of political actions "per topic" that influenced the USA during Trump presidency or a study with more relevant samples.

6.13 Next Steps

For the future developments I would like to suggest some possible areas of DL architecture that could potentially improve the current performances.

1. Working with intraday data: It appears that profit opportunities seem to have a stronger signal in the short term; therefore it is worth analyzing the effect of these methods in low latency scenarios [1]. Big Data, in fact, can be an important source of real-time predictions in our age, especially for their relative low acquisition cost [1].

2. Increasing the database dimension.

3. Implement a so-called "Attention Mechanism" that showed consistent accuracy improvements in DL related research papers on various applications.

4. Convolutional Neural Networks: They are a DL architecture that gave outstanding results with image problems and there are some studies that show they good performances in NLP problems too.

5. Recurrent Neural Tensor Network: This DL structure has the benefit of being able to grab the tree-like structure of sentences, which could translate in a deeper analysis of the context.

6. Weights initialization: As mentioned, weights are initialized randomly. In the current state, I did not modify the standard initialization distribution that the Keras API uses, but in theory there are many distributions[23] that may be worth a try.

7. Bayesian Optimization: Implementing a machine learning model whose job is to improve the current DL hyperparameters' structure would be of great help in speeding up the testing process.

8. Other forms of regularization: other than dropout, there are other methods that can be used to penalize the neural network. Such penalization forms are called regularizers[24] and can be applied both to the Input layer and to the Hidden layers. They come in different forms (the most common are L1 and L2 regularization) and as any parameter in a NN, they can be learned.

9. A more accurate analysis could be pursued by applying more evaluation metrics. Either by finding a more efficient loss function than Cross-entropy or adding other elements next to the Accuracy indicator.

[23] https://keras.io/initializers/
[24] https://keras.io/regularizers/

Chapter 7

Conclusions

The level of complexity that has been faced in this classification model is greater than in traditional sentiment analysis problems and this may explain the low accuracy presented in out model. Unfortunately, the closer we get to situations in which human discretion is involved, the more unstable a model becomes [1].

In fact the algorithm goes through lots of subjectivity [1]: What news should we consider relevant? To what extent does the writing style of a journalist matter? How many news per day should be take into account? At this point, we need to do some final considerations from both an economical and historical perspective.

7.1 The Economics behind AI

If we look closely, the objectives of traditional statistics are different from those of ML: while the first highlights being correct on average, the second cares more about operational effectiveness[1] [1], therefore ML can, to obtain more accuracy, admit some level of bias.

In fact, there is no single correct answer to what is the best DL structure to use, simply because these models have many trade-offs: more data = more accuracy, more speed = less accuracy, more autonomy = less control [1]. Machines and humans have indeed very different strengths and weaknesses [1]:

- Computers outperform us when the amount of data is overwhelming, while we make better predictions with small amount of information.
- Algorithms benefit from economies of scale[2], while we do not.
- Machines are better at predicting routine events, while humans are better at dealing with exceptional situations.

[1] which means being useful in practical scenarios

[2] which means that the more stuff we need to predict the more the unit-cost per prediction decreases.

Any company that approaches AI for creating trading strategies hopes, ultimately, to both get rid of human errors and shrink costs by automatizing as many aspects of its work-flow as possible [1]. If we look at the tasks that are most likely to be transformed first is where:

- Many parts of the process have already been automated.
- High speed of action in response to prediction yields adds value.

As you can see a trading company might have a high incentive to invest in such technology, especially by automatizing trading decisions even more [1].

Forecasting consists in grabbing data that we have, to generate information we do not have: it is a field that economics has been studying for years and perhaps by combining the old logic of decision theory and this new DL tool we can have a better understanding of what is really going on with this technology from an economic perspective [1].

The benefit of DL is that it does not require a lot of human effort nor intuition to provide outputs that were previously considered a human exclusive. In fact, the innovation of this technology is not the math behind it, that existed for decades, but the results that can be achieved by combining that knowledge with the Big Data that we have access to nowadays [1].

AI did not bring us more intelligence, but more prediction capacity; in fact, economics suggests that the drop in the cost of automatic prediction will reduce the value of its substitutes such as human prediction:

By overtaking certain tasks, AI might indeed increase competition among humans by lowering wages and disproportionately enhance the productivity of highly skilled workers [1]. Thus, it is obvious that the introduction of such tools does not cause an issue in wealth creation, but rather of its distribution among the workforce. On the other hand, AI can increase the value of complements of the decision-making process [1], namely:

1. Data.
2. Judgment.
3. Action.

Data

Gathering data is an important investment that needs be taken into account when choosing how valuable an increase in accuracy is by considering both the statistical and economic perspective of our problems [1]. On the one hand, data have diminishing returns, which means that any additional information improves your prediction less than the

previous ones. On the other hand, adding more data to a big database may be more effective than doing it to a small one, especially if this helps to pass a threshold that we care about (from an unusable model to a usable one or from worse to better than a competitor) [1].

Another related concern is the following: since DL models perform better with more data, some jurisdictions that created an environment that strongly protects their citizens' privacy, like Europe, may create a market environment where firms are less competitive globally [1].

Judgment

Decisions are usually taken under conditions of uncertainty: We decide to bring an umbrella because we think it might rain outside, but we could be wrong and have an unnecessary burden with us. Therefore, we evaluate the payoffs for every possible action we might decide to take [1].

Judgment is a process that takes time, trial-and-error and effort since it means understanding the relative payoffs linked to our actions in a certain environment [1]. The introduction of AI tools may, in fact, change the value linked to certain skills, for example coding, and enhance the overall productivity [1].

Action

Action is the process of doing something when facing a problem or a hardship. The introduction of AI might not increase the value of all indistinguishable actions, such as executing a trade, since those are most likely already automatized, but rather "high skilled" ones [1].

7.2 What can we expect in the future?

At this point there is one last question to answer: what effect could the introduction of this technology have?

Traditional economics says that by eliminating every obstacle to trading is the best way to pursue market efficiency. This translates in:

1. Deregulation.
2. Financial Innovation.

And it makes sense right? on the one hand we are removing every legal barrier that prevents people from exchanging and on the other hand new instruments, such as derivatives

and high frequency trading (HFT), are the tools to achieve it. Human imagination is the only limit! [6] This shining success pushes society to explore even more options spiralling towards the limit of no transaction costs and complete markets [6]. At the sight of all this, many economists are totally convinced that the financial market is more efficient and stable than ever before [6].

Let's now take a step back and analyze the situation, the market is in fact a really complex ecology of interacting players, all with their own strategies [6]; people often believe that by pure collaboration they will be much better off, but in reality, it is quite the opposite [6]. Have heard of "herd behavior[3]"? Social influence has the effect of pushing crowds (professionals or not) towards an inaccurate approximation due to a narrower range of opinions [6].

The "Quant meltdown" of August 2007 is a great example of this phenomenon. Hedge funds normally borrow money to increase their profits and attract investors. At some point, the strategies used by hedge funds became too similar; since there was more money chasing the same opportunities, the margin was inevitably reduced and to keep being appealing as more capital flows in...managers decided to slowly increase the leverage for years [6]. Once this race is on, no one has a choice anymore, to maintain the level of returns that investors loved so much the competition can either follow or declare bankruptcy. The result was an arms race for higher leverage: the same strategy that a drug-user chooses by injecting higher doses every time [6].

This increase in competition led to a drop in volatility [6] and with it, everything looks just great! Unfortunately, this calmness came at the expense of more frequent catastrophes: this illusion of increasing stability was a sham that led to one of the worst drops that the stock markets have ever experienced [6].

Like an upright pencil, the markets were facing an unstable equilibrium: a state that can exist if untouched, but can change instantly by receiving a even little shock [6].

The High Frequency Trading case

As we speak, HFT has undoubtedly improved the stock market quality:

1. Trading costs are lower.
2. Liquidity increased by reducing bid-ask spreads.
3. Discrepancies are minimized.
4. Prices can incorporate new information faster than ever.

[3]Herd behavior or group thinking is the situation when individuals start behaving in the same manner without centralized direction.

When it comes to trading, a few things are more attractive than velocity and companies thrive towards "zero latency": the situation where the time between the push of the button and the execution of its trade is null [6].

One question arises, is it always a good idea? It makes sense for developers to design their algorithms to allow them to escape quickly if things start going in the wrong direction. It is true that HFT increases liquidity in calm times, but they do the exact opposite in troubled ones, exactly when the market would need them the most [6].

As we have seen in this study, even complex models can predict the market behaviors accuracy levels that are slightly higher than the flip of a coin. Markets and economies indeed are among the most unpredictable and complex things that exist [6]. If you think about it, at least atoms cannot decide to change direction by their own free will [6].

The Derivatives case

The last few decades were dominated by the belief that financial innovations such as derivatives could allow the markets to reach an utopic level of stability and efficiency [6]. However Warren Buffett in a famous sentence stated: "derivatives are weapons of mass destruction" [6].

Paradoxically derivatives, passed a threshold, instead of sharing risk seem to cause more harm than good by concentrating risk in the hands of a few institutions [6]. Thus, despite lowering the risk of individual banks, they highly endanger the system as a whole and the recent global crisis has been a dramatic evidence of this [6].

Learning from the past

Efficiency means doing more with less, while stability implies the exact opposite, which is having a buffer, some extra room to absorb a hit. In fact, excessive efficiency can compromise stability [6]. The market is like a race car that starts to vibrate at excessive speed, while leverage is your foot on the accelerator: you are surely going faster, but the more time you keep this up, the more chances there are of you to cause real damage [6].

The "Positive feedback Loop" is an process by which small variations accumulate in a system and have increasingly large consequences. Why is this concept important? because markets seem to routinely behave in similar ways both in a positive and negative manner by being heavily driven by perceptions and expectations [6].

The belief that the world is becoming more efficient and stable than ever before under every aspect [6] seems to be just misleading. However, by acknowledging this we might not only be able to explain why we always seem surprised when such crises appear; but it can also help us avoiding the "this time is different" effect [6]. After all, "recognized

ignorance is better than deluded certainty" [6].

In addition, given the current hype for AI, these interconnected markets will have access to new ways to invest at incredible speed. In such context the hypothesis of some sort of "splash crash" ranging across many asset classes does not seem impossible anymore [6]. Especially if we consider that DL techniques may be particularly prone to the "herd behavior", given the fact that these algorithms need to be trained on lots of news data that may turn out to be the same [6].

To conclude, these technologies are surely interesting, but acknowledging their limitations can help us find the best ways to reduce them. The global financial crisis has revealed the need to drastically rethink how we regulate the financial system, in order to be prepared when the next recession will strike [6].

Fragility cannot be wiped out, only contained by continuously questioning our behaviors, certainties and prejudices about what can and cannot happen. The fight is rapidly shifting from humans towards machines, but they are playing games with real businesses and people and "uncertainty is no place for inaction" [6].

7.3 Related work

The sentiment analysis problems and proposed solutions to it have existed as long as written texts have. Researchers have realized that to make sure that machine could effectively pursue this task, complex model should be implemented. Sohangir 2018 [29] has been major source of inspiration both for its exhaustive analysis on the topic of stock market sentiment and for their use of deep learning as a tool to achieve it.

My analysis has been mostly focused on the possibility of linking a Deep Learning model directly to a labeled dataset rather than comparing various techniques. Since in most of the literature I referred to there was a strong favor towards recurrent neural networks those are the ones I focused on the most. I am well aware that Convolutional Neural Network, which became popular for image-related tasks have started being used to solve text-related problems too; however, this might be object of further research.

Other various studies, such as Pröllochs 2015 [25] and Ruiz-Martinez 2012 [19], focused on the difference in a way that assumes that a sentence with a positive meaning corresponds to a positive market sentiment and therefore label their datasets accordingly.

It is clear that such results may be misleading for a trading company that wants to profit from such sentiment. This is due to the fact that linguistically negative sentence does not necessarily translate into a stock market decrease; a direct system of labeling between the asset the news considered would output more practical results.

Despite this Pröllochs 2015 [25] achievements on how to catch complex negative sentences more efficiently and its useful comparison between RBS and machine learning models is certainly fascinating. However, it is necessary to note that, by dropping the assumption I have just criticized also the RBS should be re-evaluated and re-tested to be assured that they outperform ML models.

Bibliography

[1] Avi Goldfarb Ajay Agrawal, Joshua Gans. *Prediction Machines: The Simple Economics of Artificial Intelligence*. Harvard Business Review Press, 2018.

[2] Thimira Amaratunga. How deep should it be to be called deep learning? 2017. https://www.codesofinterest.com/2017/04/how-deep-should-deep-learning-be.html.

[3] Eric D. Brown. *Analysis of Twitter messages for Sentiment and Insight for use in stock market decision making*. Amazon Digital Services LLC, 2016.

[4] Jason Brownlee. A gentle introduction to exploding gradients in neural networks. 2017. https://machinelearningmastery.com/exploding-gradients-in-neural-networks/.

[5] Jason Brownlee. How to get reproducible results with keras. 2017. http://www.linguistics.fi/julkaisut/SKY2014/Wikstrom.pdf.

[6] Mark Buchanan. *Forecast: what physics, meteorology and the natural sciences can tell us about economics.* Bloomsbury, 2012.

[7] Vitaly Bushaev. Adam — latest trends in deep learning optimization. 2018. https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.

[8] Heng-Tze Cheng. Wide deep learning: Better together with tensorflow. 2016. https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html.

[9] Michael DelSole. What is one-hot encoding and how to do it. 2018. https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179.

[10] Rob DiPietro. A friendly introduction to cross-entropy loss. 2016. https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/.

[11] Sepp Hochreiter Djork-Arné Clevert, Thomas Unterthiner. Fast and accurate deep network learning by exponential linear units (elus). page 14, 2016. Research Paper.

[12] Firdaouss Doukkali. Batch normalization in neural networks. 2017. https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c.

[13] Győző Gidófalvi. Using news articles to predict stock price movements. 2001. https://www.researchgate.net/publication/228892903_Using_news_articles_to_predict_stock_price_movements.

[14] Daniel Godoy. Hyper-parameters in action! part ii — weight initializers. 2018. https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404.

[15] Yoav Goldberg. *Neural Network Methods for Natural Language Processing*. Morgan Claypool publishers, 2017.

[16] Alessio Gozzoli. Practical guide to hyperparameters search for deep learning models. 2018. https://blog.floydhub.com/guide-to-hyperparameters-search-for-deep-learning-models/.

[17] Shubham Jain.

[18] Christopher D. Manning Jeffrey Pennington, Richard Socher. Glove: Global vectors for word representation. 2014. PhD dissertation.

[19] Francisco García-Sánchez Juana María Ruiz-Martínez, Rafael Valencia-García. Semantic-based sentiment analysis in financial news. 2012. http://ceur-ws.org/Vol-862/All.pdf#page=44.

[20] Ayoosh Kathuria. Intro to optimization in deep learning: Vanishing gradients and choosing the right activation function. 2018. https://blog.paperspace.com/vanishing-gradients-activation-function/.

[21] Muriel Macdonald. How hashtags changed the way we talk. 2015.

[22] J.B. Maverick. The weak, strong and semi-strong efficient market hypotheses. 2018. https://www.investopedia.com/ask/answers/032615/what-are-differences-between-weak-strong-and-semistrong-versions-efficient-market-hypothesis.asp.

[23] Frank Millstein. *Natural language processing with python: Natural language processing Using NLTK*. Packt Publishing, 2017.

[24] Assaad Moawad. Neural networks and back-propagation explained in a simple way. 2018. https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e.

[25] Dirk Neumann Nicolas Prollochs, Stefan Feuerriegel. Enhancing sentiment analysis of financial news by detecting negation scopes. 2015. https://www.researchgate.net/publication/306059942_Enhancing_Sentiment_Analysis_of_Financial_News_by_Detecting_Negation_Scopes.

[26] Richard Nordquist. Syntactic ambiguity. 2017. https://www.thoughtco.com/syntactic-ambiguity-grammar-1692179.

[27] Christopher Olah. Understanding lstm networks. 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[28] Tal Perry. Deep learning the stock market. 2016. https://medium.com/@TalPerry/deep-learning-the-stock-market-df853d139e02.

[29] Anna Pomeranets Sahar Sohangir, Dingding Wang and Taghi M. Khoshgoftaar. Big data: Deep learning for financial sentiment analysis. 2018. https://link.springer.com/article/10.1186/s40537-017-0111-6.

[30] Kanchan Sarkar. Relu : Not a differentiable function: Why used in gradient based optimization? and other generalizations of relu. 2018. https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec.

[31] Sagar Sharma. Epoch vs batch size vs iterations. 2017. https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

[32] Andrei Shleifer. *Inefficient Markets: An Introduction to Behavioural Finance.* Clarendon Lectures in Economics, Oxford University Press, 2000.

[33] Richard Socher. Recursive deep learning for natural language processing and computer vision. page 204, 2014. PhD dissertation.

[34] Techopedia. Definition: Sentiment analysis. 2018. https://www.techopedia.com/definition/29695/sentiment-analysis.

[35] Jalaj Thanaki. *Python Natural Language Processing: Advanced machine learning and deep learning techniques for natural language processing.* CreateSpace Independent Publishing Platform, 2018.

[36] Avinash Sharma V. Understanding activation functions in neural networks. 2017. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[37] Peter Wikström. srynotfunny: Communicative functions of hashtags on twitter. 2014. http://www.linguistics.fi/julkaisut/SKY2014/Wikstrom.pdf.