Deep Learning for Sarcasm Detection on Twitter

Ostos Balfagón, Rubèn

Curs 2019-2020

Director: Horacio Saggion

GRAU EN ENGINYERIA INFORMÀTICA

Universitat Pompeu Fabra Barcelona Escola Superior Politècnica

Treball de Fi de Grau

Deep Learning for Sarcasm Detection on Twitter

BACHELOR'S THESIS

Rubèn Ostos Balfagón

Thesis Director: Horacio Saggion

Bachelor's Degree in Computer Engineering

Course 2019-2020

upf. Universitat Pompeu Fabra Barcelona

Escola d'Enginyeria

Acknowledgments

I would like to thank my family, specially my mother and my father for their support and help to achieve my goals.

To Maria, for always being there and accompanying me in this journey.

And to my tutor Horacio, for his knowledge and guidance in order to complete this project.

Agraïments

Voldria agrair a la meva família i en especial a la meva mare i al meu pare el seu suport i ajuda per tal d'assolir les meves metes.

A la Maria, per estar sempre allà i acompanyar-me en aquest viatge.

I al meu tutor Horacio, per la seva guia i coneixements per a poder completar aquest projecte.

Abstract

Automatic sarcasm detection has come very helpful to improve sentiment analysis, being proved their direct relationship recently. Numerous deep learning approaches have been applied lately in this task obtaining very interesting results.

It is common to find sarcastic messages in social networks, Twitter for example has a good API that comes handy for natural language processing tasks. From the point of view of companies and organizations it is interesting to improve their sentiment analysis models in order to have a better understanding of possible clients or even society.

The objective of this project is to analyze the most notorious work in this task up to the moment and test some deep learning architectures with different word embeddings, giving a lead on how to detect sarcasm on Twitter with good results using the Python library TensorFlow.

Keywords: Sarcasm, Deep Learning, NLP

Resum

La detecció automàtica de sarcasme ha estat de gran ajuda per a millorar l'anàlisi del sentiment, tasques que s'ha demostrat que tenen una relació directa. Nombrosos enfocaments d'aprenentatge profund s'han aplicat en aquesta tasca durant els últims anys obtenint resultats molt interessants.

És habitual de trobar missatges sarcàstics a les xarxes socials, on per exemple Twitter té una bona API que és útil per a tasques de processament de llenguatge natural. A les empreses i organitzacions els pot interessar millorar els seus models d'anàlisi de sentiments per tal d'enriquir la seva forma d'entendre a potencials clients o a la mateixa societat.

L'objectiu d'aquest projecte és el d'analitzar el treball més notori d'aquesta tasca fins a dia d'avui i provar algunes arquitectures d'aprenentatge profund amb diferents modelitzacions de paraules, donant un primer plantejament per a encarar la detecció de sarcasme a Twitter amb bons resultats mitjançant la biblioteca de Python anomenada TensorFlow.

Paraules clau: Sarcasme, Deep Learning, NLP

Prologue

Along the past years I've been interested in artificial intelligence, and after taking as much subjects related to the matter, I became mainly interested in deep learning and its numerous applications. I took the Deep Learning subject, but it was focused on image-based tasks, and although it provided good theoretical knowledge, I wanted to learn how to apply it correctly for natural language processing tasks. Following the same pattern I tried to take as much natural language related subjects as I could, learning about word embeddings and their applications.

When I was looking for a topic for my bachelor's thesis I came up with Horacio Saggion proposal for sarcasm detection on social networks, which after some research became centered around the recent deep learning approaches that have been taken regarding sarcasm detection on Twitter.

Table of Contents

1. Introduction

The Free Dictionary [34] defines sarcasm as a cutting, often ironic remark intended to express contempt or ridicule. Although it is present in our everyday life, sarcasm is not easily noticeable, and can be hard to be detected by some individuals. It has been studied over speech and linguistics, and it is finally being investigated in the fields of Artificial Intelligence and Computational Linguistics to be detected in texts, where the young task of automatic sarcasm detection is starting to find its way.

Sarcasm is closely related to irony, in fact, it is a form of irony. There is much confusion between them: traditionally, the distinction between the two was that irony was indirect, whereas sarcasm was direct, i.e. that irony involved a meaning opposite to the literal interpretation [7].

Irony is when something appears to be or is said to be one way, but it is actually another. This is often used for drama or for comedy. There are three kinds of irony: verbal, situational, and dramatic [28]:

- Dramatic irony happens when someone, such as the audience, knows that the situation is not what it appears to be, but another person, in this case an actor in play, is unaware. For example, in Macbeth by William Shakespeare, Macbeth appears to be loyal to Duncan, but she is actually plotting his murder. Duncan doesn't know Macbeth's plans, but the audience knows what is going to happen.
- Situational irony is a literary technique in which what is written or stated is different from or the opposite of what is expected. It is used for example at “The police station gets robbed” or “The fire station burns down”.
- Verbal irony is when someone says something, but they actually mean the opposite.

Sarcasm then is when someone uses verbal irony with the intent to insult or ridicule. We then find that all instances of sarcasm are irony, but not all instances of irony are sarcasm.

As Elisabeth Camp defines it at her paper in 2012, sarcasm is the expression of a dissociative attitude toward an evoked thought or perspective. She distinguishes four subclasses of it [29]:

- **Propositional**: there is a proposition that has an implicit sentiment involved. If the context is not known, it will be probably interpreted as non sarcastic.
    - For example: “*I love going to the dentist!*”.
    - In this sentence, the speaker implies they do not enjoy visiting the dentist by sarcastically expressing they do, if any listener were to read this sentence without any context, like knowing the author does not enjoy their visits to the dentist, they could imagine the author indeed loves going to the dentist.

- **Embedded**: where sarcasm is embedded in complex constructions, and is part of a compositional process. It is one of the most common forms of sarcasm.
    - For example: "*Because he's been such a fine friend, I've struck him off my list*".
    - In this sentence, the author expresses how a friend of theirs has behaved wrongfully and gained to no longer be considered a friend by saying the opposite, as part of a bigger construction with supportive phrases that lead to consider it a sarcastic expression.
- **Like-prefixed**: in this case, the sarcastic phrase starts off with "like" or "as if" and contains an implied denial of the stated argument.
    - For example: "*Like that's a good idea.*"
    - In this sentence, the author expresses sarcasm in a like-prefixed way by saying exactly the opposite of what they think, which is expressed in a particular way and started with "like", which leads to consider they are being sarcastic.
- **Illocutionary**: this kind of sarcasm involves non-textual clues that indicate an attitude opposite to a sincere utterance. It functions to evoke or allude to a certain situation in which their sincere utterance would be appropriate, but has not really happened. These utterances serve to express an evaluative attitude toward the actual circumstances that took place. It can be hard to locate and can be seen merged with metaphors.
    - For example: "*Thanks for holding the door.*" or "*How old did you say you were?*".
    - In these sentences, the author implies that the facts they are highlighting to be remembered or happened have in fact not happened, while remembering to the reader that it did not happen, in a mean way to communicate to the audience that these actions had to happen and did indeed not.

Examples above have been taken out or inspired from Reference [28].

1.1. Objective

Detecting sarcasm in social network messages could prove useful for a variety of objectives. From a marketing point of view, it could help improve a sentiment analysis campaign towards a certain company or product by detecting the sarcastic responses that did not have to be considered as positive/negative and are fact the opposite of what they were initially thought to be. It could also help detect the social reception of a political campaign or decision by the same means. The usages of sentiment analysis are widely known and have been applied over the last years, but sarcasm has been an issue to improve its results since the start of it.

The sarcasm detection task in texts has been an important tool to improve sentiment detection analysis and could see its advances applied in humor generation or detection, sarcasm generation, or any future work to be exposed. Sarcasm detection is heavily dependent on the context of the statement, and sometimes even human beings are not able to detect the underlying sarcasm in it, which could even be partially fixed some day with the help of sarcasm detection models, or at least help to.

The main objectives are to explore the state of the art of Sarcasm Detection and how Deep Learning (DL) networks work, specifically Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), in order to evaluate the usage of DL for Sarcasm Detection tasks. Since the most recent work and state of the art has been produced using Machine Learning's (ML) techniques that involve DL methods, it also serves as a way to improve the author's knowledge of said DL algorithms in a realistic practical environment.

In order to do so, this project will be developed with Python 3.8.3., using its library TensorFlow 2, which contains already a large amount of functions and tools in a comprehensive, flexible ecosystem. The code is publicly available at GitHub[1].

1.2. Structure of the document

This document is structured as follows: Section 1 is an introduction to the project and the concepts of sarcasm and irony, which is already presented. Section 2 contains a state of the art regarding automatic sarcasm detection, starting at the beginning up to DL techniques, with a conclusion highlighting issues and milestones. In Section 3 the different datasets that are used in this project are introduced. Section 4 introduces text modeling techniques with the kind of word embeddings that will be used in this project. In Section 5 the DL architectures used in this project are introduced, with a brief introduction to the key concepts of each. Section 6 contains the experiments carried out at the project, with the different accuracy and loss values obtained for each test, the results from testing some phrases in order to get predictions gave and a discussion of what we can extract from both. Finally Section 7 contains the conclusions that can be taken from the whole project. A list of references (Section 8) can be found at the end of the document, as well as an appendix with some figures.

1 https://github.com/rostos/sarcasm-tfg

2. State of the Art

The task of automatic detection of sarcasm in texts has been widely studied over the last decade, seeing an improvement in accuracy and results during the last few years. There have been many techniques applied and developed to investigate sarcasm, which has been well-studied in linguistics, psychology and cognitive science before, but has resisted its way to finally be automatically detected in texts.

2.1. Statistic and Machine Learning Approaches

This section includes statistical sarcasm detection approaches that use a set of features and approaches that use machine learning techniques in order to decide if the selected text is sarcastic or not.

As González-Ibáñez, Muresan, and Wacholder [4] point out in their paper in 2011, automatic sarcasm detection has its origin in 2006, when Nigam and Hurst [1] and Pang and Lee [2] considered the problem of sarcasm detection when they were working on sentiment polarity detection in a specified topic.

After being an issue to be considered for some years, an interesting approach was taken at Davidov et al. [3] in 2010, whose objective was to identify sarcastic and non-sarcastic utterances in Twitter tweets and in Amazon product reviews. Afterwards, the paper from González-Ibáñez et al. [4] has to be considered, where they take the first and most interesting approach to be seen in the task of sarcasm detection at the time, aiming to distinguish sarcastic Tweets from non-sarcastic ones. They built a dataset for their analysis consisting of 900 tweets (the maximum tweet length up until late 2017 was of 140 characters, when it was doubled), labeled up as sarcastic, positive or negative and picked up based on the hashtags they contain, i.e. #sarcastic for sarcasm, #happy, #joy for positive or #sadness, #angry for negative. The selected tweets must contain the hashtags they have at the end of the message, concern that was addressed before at the paper from Davidov et al. [3], that points out tweets with hashtags are noisy, thus being necessary to only consider as sarcastic the tweets with hashtags at the end, and not in between. Their approach consisted in obtaining two different groups of features (lexical and pragmatic), to then classify the tweets using a Support Vector Machine (SVM) with two different algorithms. Their accuracy was between 57% and almost 70% based on the capacity of the model to discriminate among the three categories (sarcastic/ non-sarcastic, positive or negative), which are levels comparable to human classification, as they remark. The paper concludes that the usage of those features are not enough to identify sarcasm.

In 2013, Riloff et al. [6] published their paper on detecting sarcasm as contrast between a positive sentiment and a negative situation. They take the idea of sarcasm in terms of contrast or “saying the opposite of what you mean” as their basis [6], and notice that although we can find papers exploring ways to detect sarcasm, there are no papers trying it as a contrast between a positive sentiment and a negative activity or state. In this paper we see an interesting approach trying to detect sarcasm using linguistic characteristics of it. The dataset used for evaluation purposes was manually annotated, for which they say it was not always easy for people to identify sarcasm in tweets. Their guidelines instructed human annotators to read isolated tweets and label a tweet as sarcastic if it contains comments judged to be sarcastic based solely on the content of that tweet. Tweets that do not contain sarcasm, or where potential sarcasm was unclear without seeing the prior conversational context, were labeled as not sarcastic, for example, in the case of "Yes, I meant that sarcastically.", it should be labeled as not sarcastic because the sarcastic content was (presumably) in a previous tweet. The guidelines did not contain any instructions that required positive/negative contrast to be present in the tweet, so all forms of sarcasm were considered to be positive examples [6]. The dataset contains 3200 samples overall, having 23% of them labeled as sarcastic by the annotators, where 3000 were used in the test set. In order to pick the elements that will be used as features to do the contrast between the positive sentiment and the negative situation, they develop a bootstrapping algorithm to learn lexicons of positive sentiment and negative situation phrases, the algorithm starts with a seed word, which will be 'love'. It will then learn to detect three kinds of phrases: positive verb phrases, positive predicative expressions and negative situations.

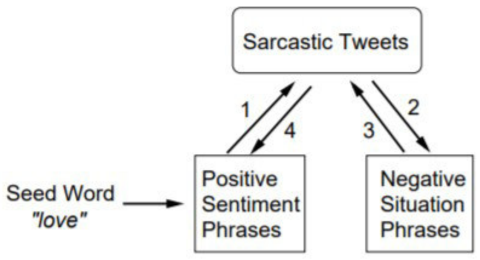


Figure 1. Bootstrapped Learning of Positive Sentiment and Negative Situation Phrases [6].

They prove that their hybrid approach using two SVM classifiers, evaluated using 10-fold cross-validation, is statistically significantly better than the SVM baselines created with the LibSVM (Chang and Lin, 2011) library that are used as baselines. Although their results are low (22% to 70% depending on the case), considering that they were detecting only one kind of sarcasm, they can be considered as successful. They point out that most sarcastic sentences carry negative sentiment.

During 2014, Maynard and Greenwood [7] published their paper investigating the impact of sarcasm on sentiment analysis. As seen before, sarcasm keeps being an issue to solve at the sentiment analysis task, which is slowly being developed to one day be applied to improve sentiment analysis methods. At the time, sarcasm is still considered too tricky to handle, and although there exist a few systems which can detect sarcasm, almost no work has been carried out on studying the effect that sarcasm has on sentiment in tweets, and on incorporating this into automatic tools for sentiment analysis [7]. In their paper, the data is extracted from a larger corpus from Preotiuc-Pietro et al. [25], the gold standard set contained 2010 hashtags and 4538 tokens taken from this corpus. They also perform a sentiment analysis of tweets to investigate the effects of sarcasm, where they pick 134 samples containing the hashtag #sarcasm. They take an interesting approach, where sentiment and sarcasm is being located by analyzing the hashtags, and in order to process them, the text analysis tool GATE is used to develop a hashtag tokenizer to find the ones related to sentiment and sarcasm easily. To do so, they extract individual tokens from multiple word hashtags and apply a set of rules that attempt to understand the different meanings and targets of the sarcasm located. Hashtags that are sarcasm indicators but not direct words that point it out are an important issue they point out, which is partly addressed i.e. negative words such as "rain" or "delay" together with a positive sentiment word such as "brilliant" [7]. For the rest, as they say, some kinds of sarcasm are incredibly hard to solve, and point out we should assume that there are some instances of it that a machine is highly unlikely to ever identify [7]. Since the paper is focused on sentiment analysis, the main results and objectives are centered around that task, where we find a section around sarcasm detection where the detection performs very well, at Table 1 we see the results, where they successfully detect most of the kind that is not indicated by any kind of sarcasm marker. It is noted that sarcasm may not always flip the polarity of a sentence.

Twitter Corpus	P	R	F1
Opinionated	65.69	77.31	71.02
Opinion + polarity	52.61	61.92	56.89
Polarity only	80.08	80.03	80.05
Sarcasm detection	91.03	91.04	91.03

Table 1. Experiments on General Tweets [7].

Besides, we can find along the years some papers developing their own frameworks, such as the one developed at Rajadesingan et al. [8], where they develop their own behavioral modeling framework called SCUBA (Sarcasm Classification Using a Behavioral modeling Approach). This framework has a unique approach by considering the past tweets of the user in order to detect sarcasm. Which by using the past 30 tweets of a user they improve the performance of their model significantly.

Bamman and Smith [9] published their paper on contextualized sarcasm detection in 2015, they notice that most of the papers on the task up to the date are text-based and ignore the potentially useful context around the tweet. In order to explain their approach, they apply the concept of the "principle of inferability", introduced by Kreuz (1996), which points out that the contextual information is only one small part of the shared common ground that must be present between a speaker and their audience in order for sarcasm to be available for use between them, leading to speakers only using sarcasm if they can be sure it will be understood by the audience, meaning that sarcasm is more likely to be used between two people who know each other well than between those who do not [9]. In this paper, the first part of the dataset contains tweets form known sarcastic accounts and has a size of 3,200 samples, which are labeled as sarcastic if they contain the hashtag #sarcasm or #sarcastic as its final term, are written in English, are not a retweet, and contain at least three words. They further sub-sample those tweets to keep only the ones that are responses to another tweet. They end up with a total of 9,767 positive and negative samples, which make for a total size of the dataset of 19,534 tweets, where the sarcastic samples are labeled also as positive. They separate between four kinds of features: tweet features, which are those scoped only over the immediate tweet being predicted; author features, those that reason over the author of that tweet, including historical data by that author; audience features, those that reason over the interactions between the original tweet user and the user replying, including historical data for both user's interaction; and response features, those that consider the interaction between the tweet being predicted and the tweet that it is responding to [9]. To compare performance across different features, they consider five feature combinations that include combinations of features with access only to tweet-level information [9], being one of the combinations of all the features together in one model. At Figure 2 we can see their results illustrating the gains in accuracy when a group of features is added, where most of these gains come simply from the addition of author information [9]. After performing an ablation test of the individual features, they find that the single most informative feature evaluated are the "author historical salient terms", for which they identify the top 100 terms in the author history of tweets with the highest TF-IDF score and create binary indicators for each of those terms. This is the single most informative feature evaluated.

Training a model on these five features alone yields an accuracy of 84.3%, less than a point behind the full feature set, that reaches 85.1%.

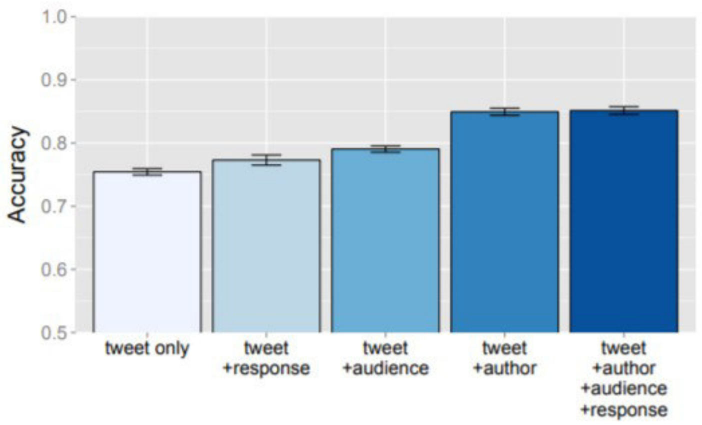


Figure 2. Accuracy across different feature sets, with 95% confidence intervals on the mean across 10 folds [9].

They conclude that, while features derived from the author yield the greatest improvements in accuracy over the tweet alone, all feature classes (response, audience and author) display statistically significant improvements over the tweet-only features that ignore the communicative context [9], confirming an improvement when context features are taken into account. They also point that the strongest audience features in order to detect sarcasm are the absence of mutual mentions, living in different cities and being popular (having many followers); all in all suggesting that the #sarcasm hashtag is not a natural indicator of sarcasm expressed between friends, but rather serves as an important communicative function for signaling the author's intent to point out sarcasm to an audience who may not otherwise be able to draw the correct inference about their message [9].

Finally, Joshi, Sharma and Bhattacharyya [10] publish their paper on context incongruity as a basis for sarcasm detection. The relationship between context incongruity and sarcasm has been studied in linguistics but never applied in depth before in automatic sarcasm detection as the basis for a model. Incongruity is defined as 'the state of being not in agreement, as with principles'. It is a necessary condition for sarcasm (Campbell and Katz, 2012). Ivanko and Pexman (2003) state that the sarcasm processing time (time taken by humans to understand sarcasm) depends on the degree of context incongruity between the statement and the context [10]. Context incongruity features applied in this paper are either explicit or implicit. Explicit incongruity appears through opposed sentiment polarity words, i.e. 'I love being ignored', which depends on the case, as it could be a normal text with different polarity contents, like a review. In the other hand, implicit incongruity is similar but the opposite polarities are expressed through phrases of implied sentiment, instead of just a single polarity word, i.e. 'I love this paper so much that I made a doggy bag out of it', where 'I made a doggy bag out of it' has an implied sentiment that is incongruous with the polar word 'love' [10]. This paper has three different kinds of datasets, where we will ignore the third since it contains discussion forum posts and not tweets. The first dataset consists of 5208 tweets, where 4170 are sarcastic, they are labeled according to their hashtags, i.e. #sarcasm and #sarcastic as sarcastic tweets and #notsarcasm and #notsarcastic as non-sarcastic tweets. The second dataset is the one Riloff et al. [6] built, which consists at the moment of 2278 tweets, 506 of them labeled as sarcastic. We have to consider that at the time, some of the tweets that were in that dataset are already deleted or unobtainable due to privacy settings, thus explaining the difference in terms of size. By taking the considerations mentioned, they build four kinds of features: lexical, which are unigrams obtained using feature selection techniques; pragmatic, which include emoticons, laughter expressions, punctuation marks and capital words and were given by the study done by Carvalho et al. [26]; explicit incongruity, which are a subset from a past system to detect thwarting by Ramteke et al. [27], and include features such as the number of times a positive word is followed by a negative word, and vice versa, or the length of the longest series of contiguous positive/negative words, among others; and implicit incongruity, which are constructed by modifying the algorithm given in Riloff et al. [6] to extract the features that will be considered implicit incongruity, although they say that the set of extracted situation phrases may contain some phrases without implicit incongruity, they hope the limited size of the tweets will prevent them to be a considerable number.

Features	**P**	**R**	**F**
Original Algorithm by Riloff et al. [6]			
Ordered	0.774	0.098	0.173
Unordered	0.799	0.337	0.474
Their system [10]			
Lexical (**Baseline**)	0.820	0.867	0.842
Lexical+Implicit	0.822	0.887	0.853
Lexical+Explicit	0.807	0.985	0.8871
All features	0.814	0.976	**0.8876**

Table 2. Comparative results for the first dataset [10].

As conclusions, they point out that an improvement in performance is achieved, showing that the usage of the relationship between context incongruity and sarcasm had success and could give an interesting output. They also point out that situations with subjective sentiment will need to be worked on.

Approach	**P**	**R**	**F**
Riloff et al. (**best reported**) [6]	0.62	0.44	0.51
Maynard and Greenwood [7]	0.46	0.38	0.41
Their system (all features) [10]	0.77	0.51	0.61

Table 3. Comparative results for their second dataset [10].

As a summary regarding statistical and ML approaches, we find an investigation of features that have been reported for statistical sarcasm detection at Joshi et al. [11], where most papers use bag-of-words. At Table 4 we can find the summarization that they have reported. For example we have González-Ibáñez et al. [4] with their sentiment lexicon-based features and pragmatic features like emoticons and user mentions; Barbieri et al. [31] which includes seven sets of features such as maximum/minimum/gap of intensity of adjectives and adverbs, max/min/average number of synonyms and synsets for words in the target text, and so on; Reyes et al. [5] with their features related to ambiguity, unexpectedness, emotional scenario, and so on; Riloff et al. [6], with a set of patterns, specifically positive verbs and negative situation phrases, as features for a classifier (in addition to the rule-based classifier); Rajadesingan et al. [8] with their uses of extensions of words, number of flips and readability features in addition to contextual features; Joshi et al. [10] are also mentioned with their features corresponding to the linguistic theory of incongruity.

	Salient Features
[4]	User mentions, emoticons, unigrams, sentiment-lexicon-based features
[5]	Ambiguity-based, semantic relatedness
[6]	Sarcastic patterns (Positive verbs, negative phrases)
[31]	Synonyms, Ambiguity, Written-spoken gap
[10]	Unigrams, Implicit incongruity-based, Explicit incongruity-based
[8]	Readability, sentiment flips, etc.

Table 4. Features used for Statistical Classifiers [11] for papers mentioned in the project.

Regarding context, three types of context are mentioned [11]:

- Author-specific context, which refers to the textual footprint of the author of the target text.
- Conversational context, which refers to text in the conversation of which the target text is a part.
- Topical context, which follows the intuition that some topics are likely to evoke sarcasm more commonly than others.

Another interesting fragment from their paper is an algorithms section which revolves around the machine learning methods used, which mostly relies on different forms of SVM, such as Davidov et al. [3] and Riloff et al. [6] with their comparison of rule-based techniques with a SVM-based classifier. There are also other approaches such as Bamman and Smith [9] using binary logistic regression.

2.2. Deep Learning Approaches

A good example of the deep learning approaches is the paper published by Mishra, Dey and Bhattacharyya [12] in 2017, where a double component CNN is used to learn cognitive features from gaze data and tweet text for sentiment and sarcasm classification. In this paper, cognitive features from the eye-movement of human readers reading the text are used, which are also referred to as gaze data, in addition to the tweets. The authors wanted to show that using a combination of automatically learned text and gaze features often yields better classification performance over CNN based systems that rely on text input alone and existing systems that rely on handcrafted gaze and textual features [12]. Although the datasets used are more complex, they have a considerable small size, being two datasets of 994 [19] and 843 [20] samples. The CNN architecture consists of two components, one for the text data, and the other for the gaze data. The text component makes use of words in the form of one-hot representation, it starts with a multi-channel variant of CNN called MultiChannelText, using two channels of embeddings: one that remains static throughout training (referred to as StaticText), and the other one that gets updated during training (referred to as NonStaticText), followed by a one dimensional convolution layer and a max-pooling for each channel. The gaze component deals with scanpaths of multiple participants annotating the same text, which are preprocessed to extract two sequences of gaze data to form separate channels of input: a sequence of normalized duration of fixations (referred to as Fixation) and a sequence of position of fixations (referred to as Saccade), an approach with both inputs is also considered, referred to as MultiChannelGaze. A two dimensional convolution is then applied that combines both inputs, to end up with a max-pooling. Once the global features are learned from both the text and gaze components, they are merged and passed to a fully connected feedforward layer followed by a SoftMax layer that will output the probabilistic distribution over the class labels [12]. The experiments are done separately for each channel and multichannel variants. The results obtained outperform traditional systems by a maximum margin of 11.27% regarding the F-score, with a maximum value of 86.97% using MultiChannelText and Fixation. The disappointing part comes when evaluating the effectiveness of the gaze data, which only improves the results by 1.34%, with an F-score of 85.63% only using text input. These points could be given with the linguistic differences in their dataset between sarcastic and non sarcastic tweets, which they argue was probably captured well by the text component. As conclusions, they discuss the results comparing the performances based on the channels used. The static channel appears to prevent over-tuning of embeddings, i.e. over-tuning often brings verbs like "love" closer to the pronoun "I" in embedding space, purely due to higher co-occurrence of these two words in sarcastic examples. For the gaze component, fixation and saccade channels perform with similar accuracy when utilized separately. They also examine how good the features learned from the CNN are, analyzing a few examples, where the addition of gaze information helps to generate features with more subtle differences for sarcastic and non-sarcastic texts. They also find that the lack of context can lead to misclassification, where the addition of gaze information does not help much in learning more distinctive features, as it becomes difficult for even humans to classify such texts [12].

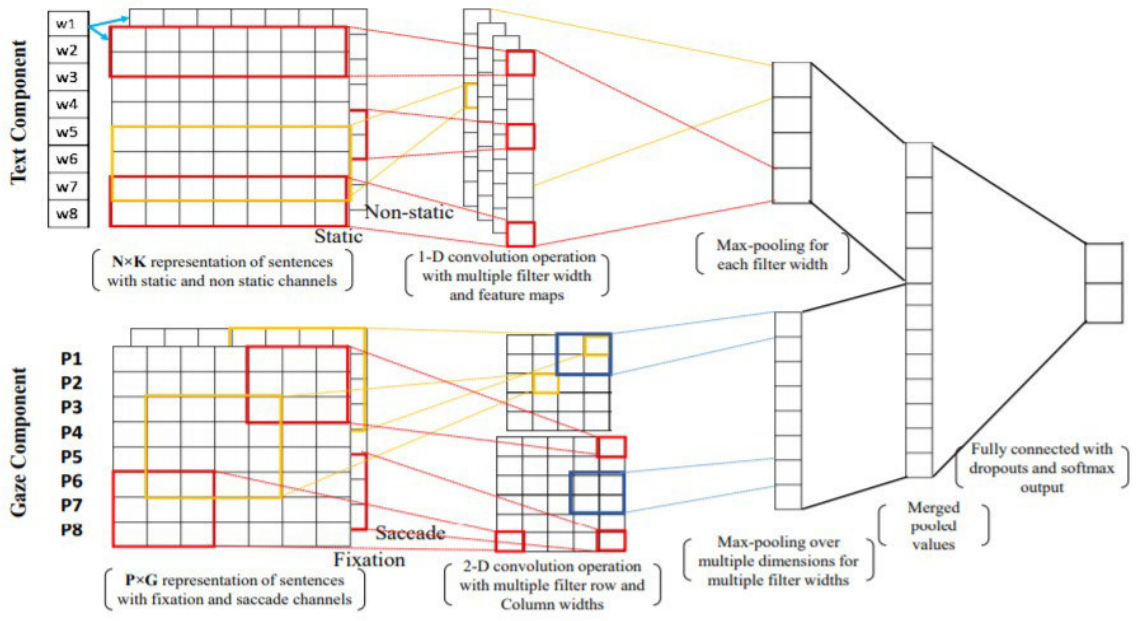

Text Component
w1
w2
w3
w4
w5
w6
w7
w8
Non-static
Static
N×K representation of sentences with static and non static channels
1-D convolution operation with multiple filter width and feature maps
Max-pooling for each filter width
Gaze Component
P1
P2
P3
P4
P5
P6
P7
P8
Saccade
Fixation
P×G representation of sentences with fixation and saccade channels
2-D convolution operation with multiple filter row and Column widths
Max-pooling over multiple dimensions for multiple filter widths
Merged pooled values
Fully connected with dropouts and softmax output

Figure 3. Deep convolutional model for feature extraction from both text and gaze inputs [12].

Afterwards, following the trend that we saw with Mishra et al. [12], we have the paper by Majumder, Poria, Peng, Chhaya, Cambria and Gelbukh [13], which takes on the task of sentiment and sarcasm classification using a deep neural network, this time taking a multitask learning approach. They argue that since both tasks are correlated, it would be interesting to consider the mutual influence between the two tasks, and point out the findings by Riloff et al. [6], that most sarcastic sentences carry negative sentiment. The same dataset we saw at Mishra et al. [12] is utilized, but this time the eye-movement data is ignored, and only the first dataset is used, which contains 994 samples. In this paper, a classifier is trained for both sarcasm and sentiment in a single neural network using multi-task learning, a novel learning scheme that gained popularity at the time. In multi-task learning, a single neural network is used to perform more than one classification task. Sentiment and sarcasm tags are taken into account, and the sentences are represented using Glove [30] word-embeddings of size 300. This procedure is being carried out on by a Gate Recurrent Unit (GRU) network.

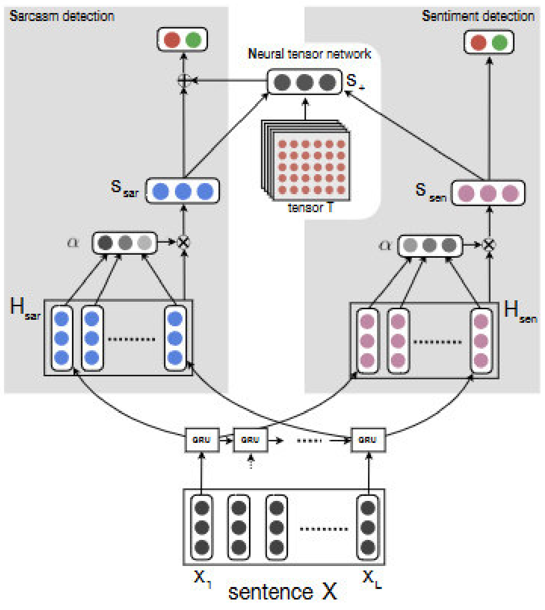


Figure 4. Multi-task architecture used [13].

After the GRU transformation of the sentences, task-specific sentence-level context is encoded to the word representations, which will be fed to a Neural Tensor Network (NTN) to fuse sarcasm and sentiment sentence representations. The output from this network will be a vector that contains information relevant to both sentiment and sarcasm, concatenating each element of the sentence. Different softmax layers are used for each kind of classification. Optimization during training will be done using the Adam algorithm, and the loss function applied will be categorical cross-entropy. This paper surpasses the state of the art at the time which was the paper published by Mishra et al. [12] in 2017, which will be their baseline, as well as the standalone classifiers for each task. They compare the results obtained with the baselines mentioned, at Table 5. The results are obtained using a 10-fold cross validation

Variant	Precision	Recall	F-Score
State of the art [12]	87.42	87.03	86.97
Standalone classifiers	89.96	89.25	89.37
Standalone coerced	-	-	-
Multi-Task simple	89.42	89.19	89.04
Multi-Task with fusion	90.94	**90.74**	**90.67**
Multi-Task with fusion and separate GRUs	**91.01**	90.66	90.62
Multi-Task with fusion and shared attention	90.50	90.34	90.29

Table 5. Sarcasm results for various experiments [13].

As conclusions, they have empirically shown that this method outperforms the results obtained with two separate classifiers and, in particular, outperforms the state of the art at the time by Mishra et al. [12]. They did so by using a GRU instead of a CNN as we saw in that paper. This paper shows that multi-task learning-based methods significantly outperform standalone sentiment and sarcasm classifiers, which indicates that sentiment classification and sarcasm detection are related tasks.

We can see how starting in 2016, there is an increase of popularity in deep learning techniques for natural language programming applications, a few such approaches have been reported for automatic sarcasm detection as well [11]. As mentioned in Joshi et al. [11], we mostly find Convolutional Neural Network (CNN) architectures, such as Poria et al. [14] investigating the usage of those approaches in sarcasm detection, there are also combinations of different models, such as Recurrent Neural Network (Long Short-Term Memory) with CNN by Ghosh et al. [15], where they compare their approach against recursive SVM and show an improvement for the deep learning architecture. The paper from Mishra et al. [12] takes another step while using a CNN architecture, while investigating the effectiveness of gaze data. The paper from Majumder et al. [13] makes use of a GRU for sentence transformation while using a custom NTN for multi-task learning, proving that sentiment analysis and sarcasm detection tasks are related and help each other improve their respective results.

2.3. Conclusions

The paper from Joshi et al. [11] highlights important findings in the task at the moment in 2016, which have barely been addressed. If we only consider the twitter dataset papers, we find conclusions such as that unigram-based features outperform the use of a subset of words as derived from a sentiment lexicon in González-Ibáñez et al. [4] or that historical features along with flip-based features are the most discriminating features in Rajadesingan et al [8]. An analysis of systems based on types of sarcasm is not available in most past work, primarily due to lack of corpora in which different kinds of sarcasm are marked [11]. Some papers take on the issue to classify into one of the four sarcasm types, where in Camp [16] it is observed that nearly 71% of the samples require context, that is, illocutionary or like-prefixed.

Some design issues that are present in the task are mentioned, which include issues with the quality of the annotations, issues in the usage of certain hashtags such as #not, which is often used to express sarcasm, while the rest of the sentence is not sufficient for identifying the sarcasm [11], a thing that can be partially fixed using author-specific context. Or in the case of manually annotated datasets, where the quality of the annotation is a concern, where it can differ based on the culture of the annotator. The above observations highlight the need of framing appropriate guidelines for annotators.

The second issue deals with using sentiment as a feature for classification. The section regarding this issue has an analysis of papers using sentiment in any way, used as a feature as the apparent polarity of a sentence. Here we can find papers using sentiment of a past tweet by the author to predict sarcasm, for example. In Maynard and Greenwood [7] it is demonstrated the impact of sarcasm detection on sentiment classification using modules from the GATE framework, where, as we have noted before, they observe that sarcasm may not always flip the polarity. Lately, the paper from Majumder et al. [13] proves that sentiment and sarcasm are directly related.

Finally, the third issue lies in the context of handling unbalanced datasets. This issue comes from the usage in certain papers of unbalanced datasets, datasets where a small set of tweets are marked as sarcastic compared to the total number of tweets in the dataset. To solve those cases, methods such as applying a multi-strategy ensemble learning approach, using SVM-perf that performs F-score optimization or an LSS-regularization strategy, besides others, are applied over different papers. It is noted that data imbalance also influences the choice of performance metrics reported [11], since AUC is known to be a more reliable indicator of performance than F-score for skewed data. In Abercrombie and Hovy [18], performance of sarcasm classification for many datasets of different data imbalances is compared, and they indicate that both class imbalance and labelling methods affect performance, and should both be considered when designing automatic sarcasm detection systems.

The paper points out three milestones in the history of sarcasm detection research at the moment: semi-supervised pattern extraction to identify implicit sentiment, the use of hashtag-based supervision to create large-scale annotated datasets, and the use of context beyond target text.

3. Datasets

We have seen how the different papers manage and acquire the data to build their datasets, but most of said datasets are commonly private or if not, they are sometimes built using tweet IDs, which, depending on the year they were collected, will lead to the dataset being severely reduced, as seen in Riloff et al. [6], where some tweets that contained are already deleted or unobtainable due to privacy settings.

There is also the labeling issue, which can be done in various ways, and is usually addressed by labeling according to the hashtags the tweet contains. As pointed out in many papers, and as Joshi et al. [11] highlight in their "Issues in Sarcasm Detection" section, if the annotations are created by a different person than the author, they can vary based on the culture of the first, which can be troublesome.

Although dataset creation is always possible, it would need for one of the following two approaches to be followed regarding the annotations: using hashtags for labeling or labeling according to an annotator opinion. It also needs for dataset purge regarding uninteresting tweets such as spam or link-based tweets, besides others. The collecting methodology would need to be clear and follow a specific pattern. The decisions taken in this step will influence the whole project.

That being said, we are able to find various public english sarcasm datasets around the Internet, with a couple of them being from published papers. Although it is common for datasets that have no paper relation to have no clear descriptions and sometimes are missing its context or basic information, making the search and usage of datasets in this project more complex, which will be addressed by using only paper related datasets regarding the training of the models. In this section an analysis and exposition of the datasets that will be used in the different parts of the project will be carried out.

3.1. Datasets used

3.1.1. Dataset 1

The first dataset will be the one used in order to do the different tests. This dataset is used in the papers from Mishra, Dey and Bhattacharyya [12] and Majumder, Poria, Peng, Chhaya, Cambria and Gelbukh [13], and was created at the paper of Mishra, Kanojia and Bhattacharyya [19].

The decision to use solely this dataset for testing comes from its background, it has been used before in two papers which were also using deep learning methods, giving it contrasted reliability. The fact that its context, creation process and information is clearly specified and publicly available makes this dataset the main asset of this project regarding the dataset section.

Having two papers that used it will provide a ground for comparison once the tests are over.

It contains two csv files, “Fixtion_Sequence.csv” and “text_and_annorations.csv”. The first is the eye-movement data that was used in Reference [12] and was mentioned in Section 2.3. (Deep Learning Approaches), it contains for each participant, for each sentence, words in the sentence are presented in the order they were fixated, along with the fixation duration in milliseconds. The second is the corpus per se, which contains the sentences taken for their experiment and annotation results, with 1000 samples. We are interested in the second csv, which contains 7 kinds of columns, the last being a group of 6 columns that contain the polarity as every participant considered. We are interested in the second column, named “Text”, which contains the sarcastic or not sarcastic sample, and the sixth column, named “Sarcasm”, which contains the sarcasm label, labeled as Yes or No. Other remarkable columns are the default polarity and participants polarity in order to use a sentiment detection model, and the source column, which gives us the origin of the text.

I work 40 hours a week to be this poor
Nice perfume. Must you marinate in it?
Suburbia: where they tear out the trees and then name streets after them.
This isnt an office. Its Hell with fluorescent lighting.
Dont bother me. Im living happily ever after.
Sometimes I need what only you can provide: your absence.
I feel so miserable without you, it's almost like having you here.
I never forget a face, but in your case I'll be glad to make an exception.
The trouble with her is that she lacks the power of conversation but not the power of speech.
Weather forecast for tonight: dark.

Table 6. First 10 samples from “text_and_annorations.csv” [19].

The text can be from sarcastic websites that include quotes or funny phrases, social media websites, which include manually extracted tweets, or from the Amazon Movie Corpus from Pang and Lee [40]. The origins of the text and the number of samples for each can be found at Table 7. The dataset contains 350 sarcastic samples and 650 non sarcastic.

Origin	Number of samples
Sarcastic quote websites	103
Amazon Movie Corpus	76
Twitter	171

Table 7. Number of sarcastic samples and their origins at “text_and_annorations.csv” [19].

As stated in their paper, two expert linguistics validated with 100% agreement that the 350 sentences extracted from various sources are indeed sarcastic and express negative sentiment towards the main aspect [19].

3.1.2. Dataset 2

The second dataset will mainly be used to obtain sarcastic phrases in order to test the different models, in the prediction section, Section 6.2. This dataset is created in the paper from Oraby, Harrison, Reed, Hernandez, Riloff and Walker [41], and is a subset from the Internet Argument Corpus from Walker, Anand, Fox Tree, Abbott and King [42], which is a corpus for research in political debate on internet forums. The dataset contains 4692 samples. It consists of one file, “sarcasm_v2.csv”, which contains dialogue data. The format consists of a quote text with a response text, the last being sarcastic or not. For this, we are interested in the second column, which specifies the sarcasm label, being “sarc” or “notsarc”, the fourth column, with the quote text, the text that the reply was given to, and the fifth column, that contains the response text. Since the dataset is not built on tweets, it could be counterproductive and unrealistic to use it for testing, its quote-response nature makes the training more complex, and it has not been used before in papers related to deep learning methods. Although it could appear that this kind of dataset is not useful for our project, it contains long sarcastic phrases, which are rare to see, and will prove useful for testing, when combined with short sarcastic phrases from the following dataset.

Quote	First off, That's grade A USDA approved Liberalism in a nutshell.
Response	Therefore you accept that the Republican party almost as a whole is "grade A USDA approved Liberalism." About time you did.

Table 8.First sample from “sarcasm_v2.csv” [41].

The discussions that the dataset contains are extracted from the websites “4forums”, “ConvinceMe” and a sample from “CreateDebate”. They were labeled using AutoSlog-TS, a weakly-supervised pattern learner that extracts lexico-syntactic patterns associated with the input data created by Riloff [43], for non sarcastic samples and with the help of Amazon Mechanical Turk for sarcastic ones. Each labeling Turker had to pass a test in order to be able to label, with a score above 70%.

Subset	Number of samples	Sarcastic/Non sarcastic
Generic	3,260	1,630
Rhetorical Questions	850	425
Hyperbole	582	291

Table 9. Number of samples at “sarcasm_v2.csv” [41].

This is a balanced dataset as it has the same sarcastic values as non sarcastic, as can be seen in Table 9, it also contains different kinds of questions, based in the kind of quote-reply relation we can observe at each sample, which are grouped in different subsets according to their category.

3.1.3. Dataset 3

The third dataset will be used as complement to the second dataset for testing the predictions that the models generate in Section 6.2. This dataset has been obtained from kaggle, and it has been updated and supposedly created by the user “akshit3050” [44]. The dataset contains a huge amount of data, with over 900 thousand unique values. It consists of two files, “data_train_sarcasm.csv” and “data_test_sarcasm.csv” which are already separated groups for training and testing and contain the same kind of data. Each record consists of the sarcasm label, represented as 1 for sarcastic and 0 for non sarcastic, and the text, named comment in the column. The dataset comes with Reddit API license, which indicates the data has been collected from Reddit, but there is no information on this, or how the labeling was carried out. This lack of information, and the fact that once again the data does not come from Twitter makes its usage for testing in the models counterproductive, without considering the fact that there is no information on if it was used for any kind of project, related to deep learning or not. But its huge size and the fact that it contains small phrases could be useful when we reach the prediction phase.

NC and NH.
You do know west teams play against west teams more than east teams right?
They were underdogs earlier today, but since Gronk's announcement this afternoon, the Vegas line has moved to patriots -1
This meme isn't funny none of the "new york nigga" ones are.
I could use one of those tools.
I don't pay attention to her, but as long as she's legal I wouldn't kick her out of bed (before she took a load)
Trick or treating in general is just weird...
Blade Mastery+Masamune or GTFO!
You don't have to, you have a good build, buy games or save it
I would love to see him at lolla.
I think a significant amount would be against spending their tax dollars on other people.
Damn I was hoping God was real
They have an agenda.
Great idea!

Table 10. First 14 samples from “data_train_sarcasm.csv” [44].

The dataset contains 904.208 unique values, taking into account the duplicate values, the dataset contains 478.451 non sarcastic values and 471.548 sarcastic values. As stated before, there is no information on how the labeling was carried out, and it appears that the data was not cleaned, considering the duplicates and the weird samples. This is common when talking about public datasets. When it comes to Twitter datasets, we can also find that the mentions or hashtags have not been cleaned out. The good aspect of this dataset is that the big amount of data gives room to choose an apparently good sample for prediction.

4. Text Modeling

We talk about text modeling when the data has to be transformed in order to be processed by the different architectures. When it is about natural language processing systems, it comes to managing the text in order to transform it to mathematical representations or extract certain features from the text words by applying mathematical operations to it.

At the start, as we saw in the State of the Art (section 2), feature engineering is applied in order to extract the information that every group of investigators that published each paper considers important, which happens on rule-based approaches and statistical sarcasm detection, and keeps being the basis for machine learning approaches. Once we reach deep learning approaches we begin to see transformations of the text to represent each word with mathematical representations, which are called word embeddings.

In this project, word embeddings are used in order to map the words of each tweet from the dataset into vectors of real numbers, leading to a transformation of each tweet into a matrix of vectors. The words can be then embedded without applying any kind of manipulation or simplification based on the suffixes or prefixes that they display, an example being the case of working, work and worked being different between them or the three words being stemmed in order to be able to be equal ignoring any suffixes that they contain. The resulting stem of the word does not have to be specifically their morphological root, being different according to the word, which may differ according to the kind of stemmer applied. In this project, Lancaster Stemmer [35] will be used for testing when stemming is applied. Since we are dealing with casual talking, an aggressive stemmer such as that will prove useful.

4.1. Word embeddings

A good example to introduce word embeddings are the "One-Hot Encodings", with that kind of representation, the different category distribution is transformed to a vector with size equal to the number of different categories, minus one considering we start at zero. This way, each word or element has a value of 1 according to the category it has, the rest of elements of the vector will have a value of 0. But that kind of word embeddings would not prove useful in our case, since we have to transform each word to a unique representation, having the big dictionary that our case occupies makes this case impossible, since we would have vectors of gigantic size affecting our performance.

Here is where the models to produce word embeddings come to use, in our case, each word needs to be transformed to a vector of a considerable size, size that could vary to test its influence in our experiments. In this project, two of the most known models will be used: Word2Vec and GloVe. They both have been applied in the papers mentioned before in the State of the Art (Section 2) and are the principal methods to produce word embeddings nowadays.

When we issue of the size of said word embeddings, if the optimal size is tried to be found by experimenting, good results are obtained with sizes from 120 to 300 based on the case, with slight variations depending on the test. In order to use the most accurate size, we find papers such as Pennington et al. [37] or articles that take the size of 300 as the optimal size for Word2Vec and GloVe word embeddings. When looking for pre-trained models, it can be established a well-known model for Word2Vec, one trained with the Google News dataset (which has a size of about 100 thousand million words), that indeed uses word embeddings of size 300. The case is slightly different for GloVe, there are four officially released and reliable pre-trained word vectors, two built on Common Crawl website data and one that uses Wikipedia data from 2014 and the Gigaword 5 [45], these three use embeddings of size 300, but the fourth is built on Twitter data, and has vectors of size 200. Since this project revolves around Twitter, it was decided to use the fourth dataset, having as reference the usage of word embeddings of size 300 for the Word2Vec tests and 200 for the GloVe tests, with the purpose to make fairer comparisons of the results from pre-trained and trained at the moment models.

Regarding these models, they can be developed before (or even meanwhile) the training of the deep learning models is realized. In this process, each word or stem starts with a vector of random values with the determined size for the vector, then the corresponding training process takes place, ending up with a vector representation for each word or stem. It could also occur that our vocabulary size was limited due to a short corpus or its specific circumstances, in that case, pre-trained models for each kind of word embeddings are available to use to test its efficiency compared to the models trained from scratch, this means that we skip the generation process for the word embeddings model and instead we load a pre-trained model. In our tests, we will compare how this affects our results, and what we can extract of each methodology and different kind of word embedding.

4.1.1. Word2Vec

Word2Vec is a group of two models with different architectures that are used to produce word embeddings, these models make use of a two layer neural network that receives as input a text corpus, and processes it in order to obtain vector representations for each word that the corpus contains. This way, deep learning networks can understand each word and process them.

It trains each word against other words that neighbor them in the text corpus, by either using context to predict a target word (a method known as continuous bag of words, or CBOW), or using a word to predict a target context, which is called skip-gram [36]. Both methods performance will be compared at the test section.

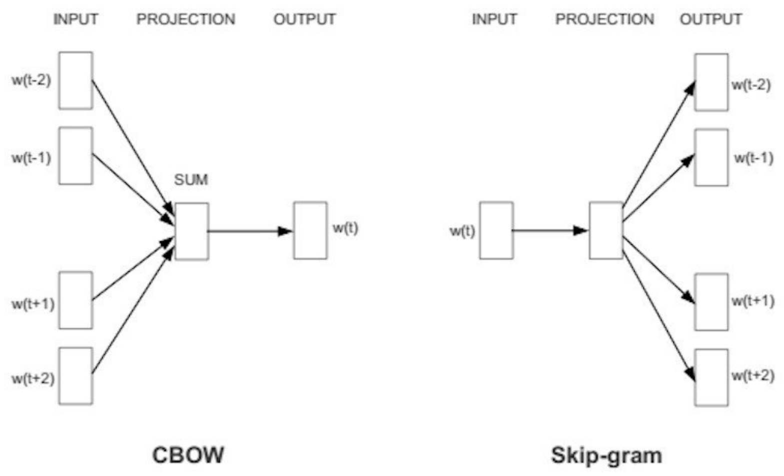


Figure 5. The two models used by Word2Vec [36].

The CBOW method starts off with an input of one-hot encoded context words that go through a hidden layer via a weight matrix, which connects to the output layer with its own weight matrix, the output is a target word leveraging all words in its neighborhood. Skip-gram has the same architecture but mirrored, which takes a single word as input in order to predict the context words given that original word. The difference in the values that the hidden layer receives comparing both methods, since CBOW takes various inputs, resides in the fact that they are calculated by summing each value that the input layer projects to the hidden layer, making an average which is weighted by the weight matrix from that layer. While in Skip-gram there is no operation to average since it is only one word. Besides the architecture, the main difference lies in the hidden layer output that is given to the output layer, which go through a softmax function. In the case of CBOW, there is one softmax, while Skip-gram has a certain number of softmax operations, based in the number of context words that it is trying to find. The process has its own forward and backward propagation steps that will fit the weight matrix for each layer.

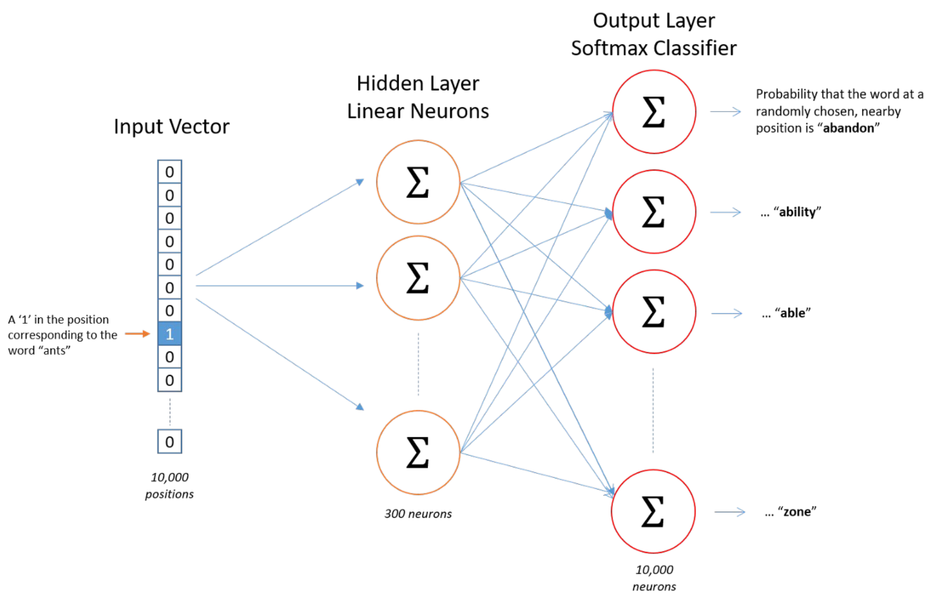


Figure 6. Neural Network for the Skip-Gram model [47].

The output results will be word embeddings, but one-hot encoded as well as the input was. We are in fact interested in the hidden layer weight matrix that each word has produced once the training is over, where in the case of Skip-gram it is a matrix of vectors. These will be the word embeddings for the word (for CBOW) or words (for Skip-gram) that our output has given.

The word embeddings then will be produced such that words with similar meaning are placed together in the word vector space. Since the size of the vectors is too high to be represented or imagined, in order to do so, the relative meanings are translated to measurable distances by applying Principal Component Analysis (PCA) projections. This way, each group of words can be represented in order for us to understand the representations. This will lead to interesting graphics, and relations between words can also be traced.

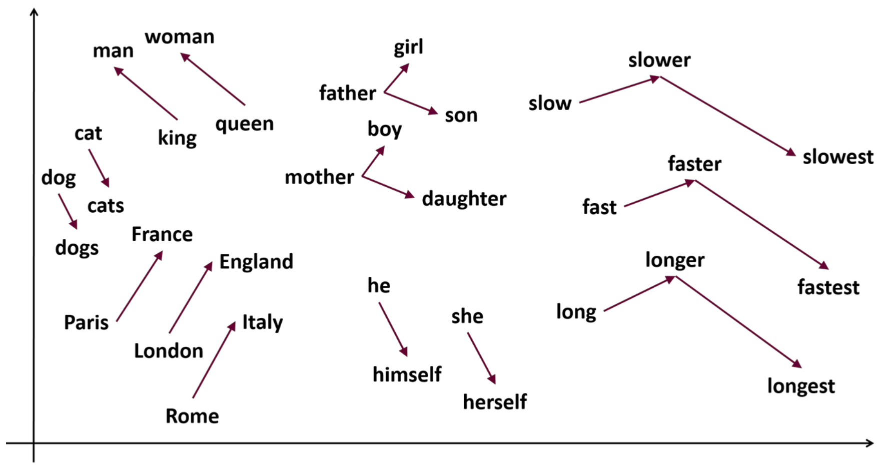


Figure 7. Semantic relations in the Word2Vec vector space [48].

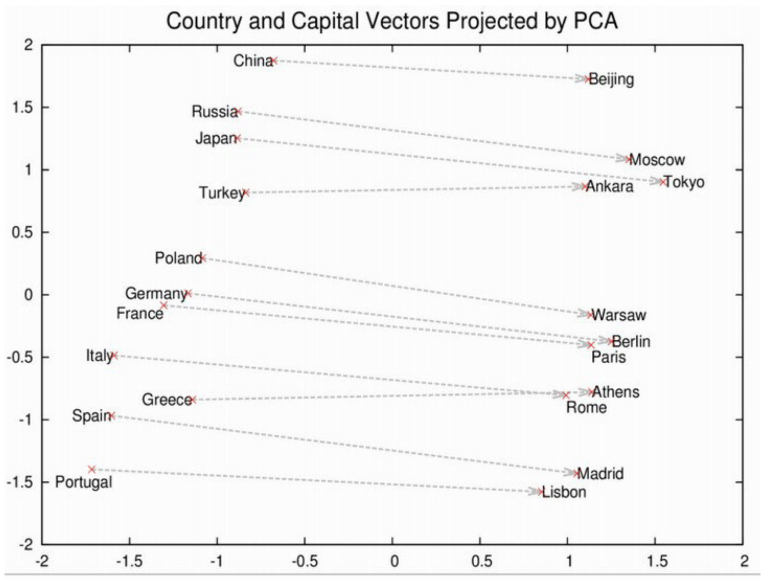


Figure 8. Vector representation for country and capital in Word2Vec [36].

Word2Vec gives us the function to obtain words that do not match in a certain group or make operations with the word embeddings. For example, an operation involving a sum of the word vectors of “king” and “man”, minus “queen” should give as result the word “woman”. The most effective results come when we try to find similar words to one or more than one word, obtaining a list of words and the grade of similarity.

Word	**Similarity**
brother	0.85411
uncle	0.84345
nephew	0.84343
grandson	0.82373
daughter	0.81417
eldest son	0.80417
mother	0.80088
dad	0.79508
stepfather	0.79165
sons	0.79084

Table 11. Code results for the 10 most similar words to father and son from the pre-trained Word2Vec from Google News.

4.1.1.1. Code implementation

At the code section of the project, Word2Vec has been implemented in two ways: by using a pre-trained model and by training the model from scratch at the moment with the available vocabulary from the corpus. Both implementations will be tested at the testing section.

4.1.1.1.1. Pre-trained

The pre-trained model has been obtained from the official Google Drive where it is updated, which link is available at the official website from Google Code. A Github repository from 3Top [46] was helpful in order to compare the different publicly available pre-trained word embeddings available at the web. As stated at the repository and Google website, the model contains 3 million word embeddings of size 300 and it was created using the Google News Dataset which has a size of 100 thousand million words. This model has been trained applying the negative sampling algorithm, which made it faster to train. There are at least 4 other different Word2Vec models available to download, but its size is considerably smaller (up to 1.4 million word embeddings) compared to Google News’ and they apparently do not provide any interesting improvement compared to it, as they are from Wikipedia or Freebase.

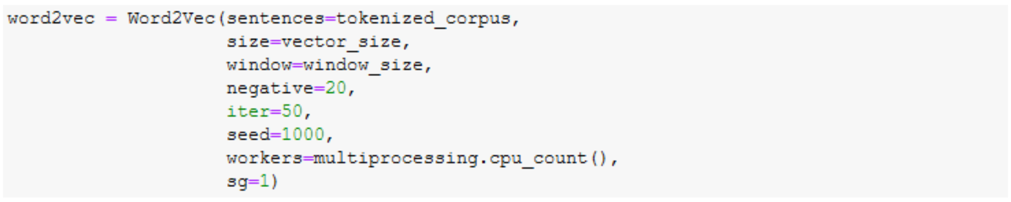
As it is stated in the paper from Mikolov et al. [63], this model has been created with the Skip-gram model, for which they mention that is slower but better for infrequent words, while CBOW is simply faster. The process to load the model is fairly simple, as it only requires for its file to be loaded. Said loading will take time and a considerable amount of memory which also loads the original corpus, we are able to ditch said corpus and pick only the word embeddings, if we want to, in order to free some memory.

4.1.1.1.2. Trained from scratch

The Word2Vec model is trained with the function that the Gensim library provides for such actions, which has its same name. It requests some parameters, which are listed below with their purpose:

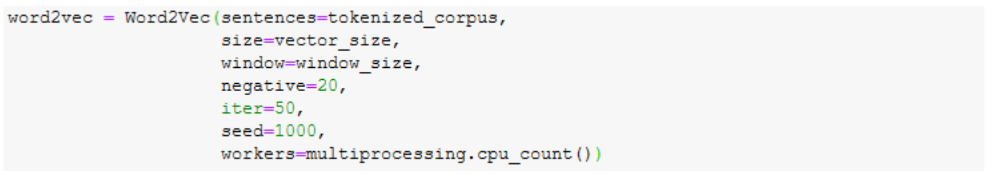
- **Sentences**: Sentences to be evaluated. The corpus separated by sentences, which in our case are each tweet as a vector in a matrix of tweets.
- **Size**: Size that the word embeddings will have. As stated above, it has been chosen to be 300, based on the paper from [37] and the fact that the pre-trained model has a size of 300 as well, in order to compare both approaches.
- **Window**: Maximum distance between the current and predicted word within a sentence.
- **Negative**: if the value is >0 it applies the negative sampling algorithm to fasten the training. Amount of "noise words" that it will use. The chosen value is 20 as it is usually between 5 and 20.
- **Iter**: number of iterations that the model will do over the corpus. Selected 50 by default since it was used in the examples consulted.
- **Seed**: Seed for the random number generator that will generate the initial vectors for each word.
- **Workers**: number of processor cores that will work on the training. At the code we obtain the total number of cores to work faster.
- **Sg**: value that determines if the Word2Vec model uses Skip-gram (value 1) or CBOW (remove the parameter or 0).

With each parameter clarified, we will test both available architectures, Skip-gram and CBOW, by doing each test for each model, results of which will be available at the test section.



```
word2vec = Word2Vec(sentences=tokenized_corpus,
                    size=vector_size,
                    window=window_size,
                    negative=20,
                    iter=50,
                    seed=1000,
                    workers=multiprocessing.cpu_count(),
                    sg=1)
```

Figure 9. Code and parameters used to train Word2Vec from scratch using Skip-gram.



```
word2vec = Word2Vec(sentences=tokenized_corpus,
                    size=vector_size,
                    window=window_size,
                    negative=20,
                    iter=50,
                    seed=1000,
                    workers=multiprocessing.cpu_count())
```

Figure 10. Code and parameters used to train Word2Vec from scratch using CBOW.

Once the function is set and ran, the logs show us how the training is performed. It gives us a glimpse of how Word2Vec works through the number of iterations that have been set. Since the vocabulary size is rather small (we are using Dataset 1 for training) compared to the size the pre-trained model has, the time it takes for training is considerably smaller than the time it takes for loading the pre-trained model.

4.1.2. GloVe

GloVe has a different way of creating the word embeddings than Word2Vec. Instead of using a neural network for that purpose, they are generated with a series of equations in an unsupervised learning algorithm. The main difference besides the methodology to achieve their word embeddings resides in the fact that Word2Vec relies in local statistics in order to create them, and GloVe incorporates global statistics. For this, they use the number of times the word appears in the context of another word, the number of times it has appeared and the conditional probability of given the first with the occurrence of the second. Using that last value, they calculate the ratio of co-occurrence of two words by dividing that third value of two different words between them. This is used to obtain the similarity (or difference in case the value is low) between words.

X_{ij} tabulate the number of times word j occurs in the context of word i.
$X_i = \Sigma_k X_{ik}$
$P_{ij} = P(j\|i) = X_{ij} / X_i$

Table 12. Basis values of GloVe [49].

By managing the ratio of co-occurrence, GloVe measures the similarity of the hidden factors between words to predict their co-occurrence count, computing these values with exponential functions. The loss function applied is the difference between between the actual observed frequencies and the predicted frequencies, a cost function is used in order to calculate the error and the predicted co-occurrence counts. But since each word pair have different occurrence frequency in the corpus, we need a weight to readjust the cost for each word pair. After several iterations in order to train the predictions, we will end up with vector representations that predict the co-occurrence counts for each word.

Pennington, Socher, and Manning argue at the GloVe paper [30] that they produce better embeddings faster than Word2Vec, this is proved through many experimental results. Empirical results published after the GloVe paper appear to favor the conclusion that GloVe and word2vec perform roughly the same for many tasks [50]. If we consider that pre-trained models are available and usually used instead of training an own model at the moment, the training time becomes irrelevant.

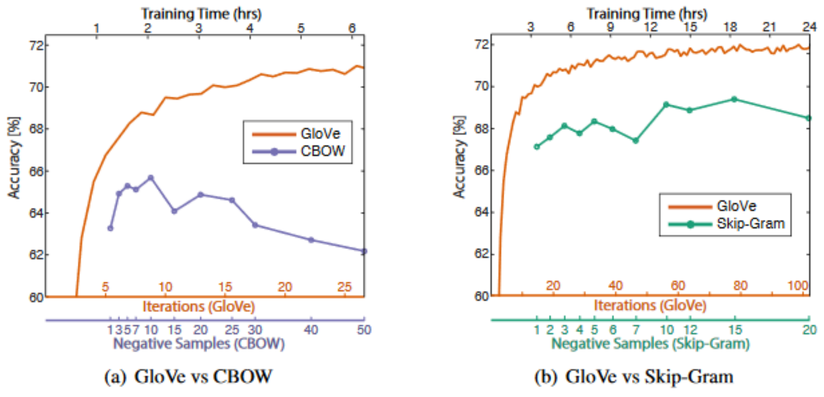


Figure 11. Overall accuracy compared to Word2Vec's models [30].

By applying nearest neighbor evaluations' similarity metric, it is possible to produce a single scalar that quantifies the relatedness of two words. This way, we are able to graphically represent relationships between words that happen in the GloVe model.

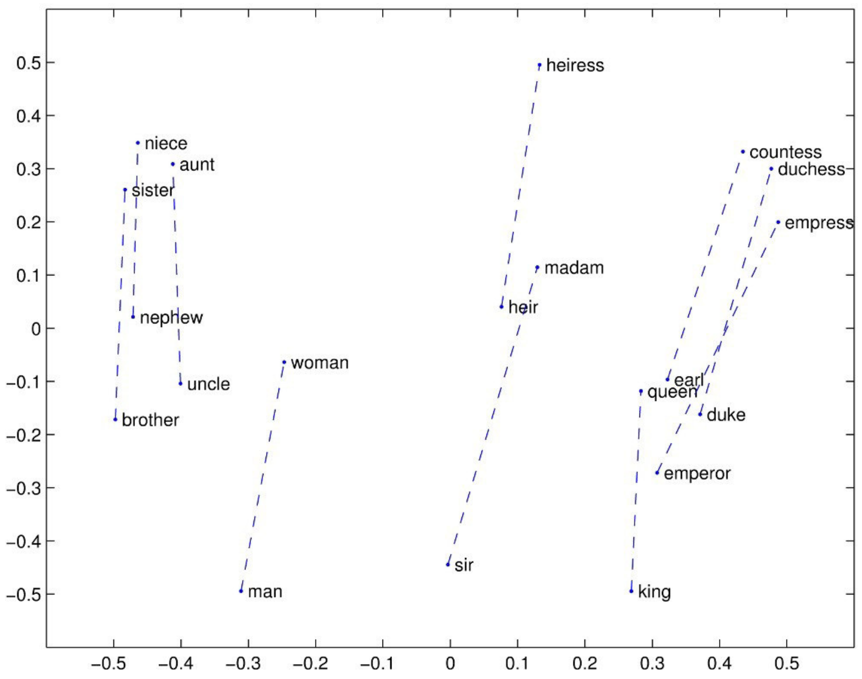


Figure 12. Visual representation of relations from GloVe word embeddings regarding man-woman pair of words [51].

The GloVe word embeddings are used as much as Word2Vec's, for example, at our State of the Art (Section 2), the paper from Majumder, Poria, Peng, Chhaya, Cambria, and Gelbukh [13] uses them with vectors of size 300, although it is not mentioned if a pre-trained model is used, it is highly probable that the word embeddings were trained from scratch.

4.1.2.1. Code implementation

At the code section of the project, GloVe, like Word2Vec, has been implemented in two ways: by using a pre-trained model and by training the model from scratch at the moment with the available vocabulary from the corpus. Both implementations will be tested at the testing section.

4.1.1.1.2. Pre-trained

The pre-trained model has been obtained from the official GloVe website of The Stanford Natural Language Processing Group [51]. As it was mentioned before, we are able to find four kinds of different pre-trained models, but since we are working with tweets, the Twitter model was chosen for this project. It comes in four different word embeddings dimensions: 25, 50, 100 and 200. Since we aim to be as close to 300 as possible, the model of size 200 was chosen. It has been trained using 2 thousand million tweets, which produced 27 thousand million tokens, resulting in a total vocabulary size of 1.2 million word embeddings. The process to load the model is fairly simple as each record in the saved model file is stored in a Python dictionary in order to lately find the word embedding for each word easily.

4.1.2.1.2. Trained from scratch

In order to generate the GloVe model at the moment, a complementary set of functions from Grady Simon [52] was used. In it, we are able to find the class “GloVeModel” to create a GloVe model from scratch as an object with the following necessary parameters:

- **Embedding size**: Size that the word embeddings will have. As stated above, it has been chosen to be 200, based on the fact that the pre-trained model has a size of 200, in order to compare both approaches.
- **Context size**: Number of tokens that will be evaluated as context for each word for each side, meaning we will have double the size of this as context.

Once the model has been created, we are able to implement the corpus to work on in order for the model to build the co-occurrence matrix that corresponds to it. Finally, the training can be executed with the number of epochs for it that we desire to. The value for the number of epochs is recommended to be between 50 and 100, where 100 was chosen in order to follow the example, which is the same case for the context size, that will be 10.

Finding a way to train your own GloVe word embeddings can be hard if we are not willing to compute them by hand or create it from scratch (which would be tremendously complex). The functions from Grady Simon have come handy and they are also easy to implement in our case along with the TensorFlow procedures.

5. Architectures

The field of artificial intelligence comprehends a series of technologies, one being the machine learning architectures, which itself contain the deep learning architectures, a category that comprehends a series of artificial neural networks. We can describe neural networks as a step to imitate the activity that our neural layers have at the neocortex in our brain, the area where thought occurs [53]. Said neural networks will learn hierarchical structures and representation levels in order to learn patterns in the data that we input to them.

The architecture structure of a neural network can be reduced to three parts:

- **Input layer**: receives the input data.
- **Hidden layers**: various layers on top of each other, each containing a certain amount of neurons that are interconnected and contain their own parameters, which transform the data they obtain from the layer behind. Its work together will able us to obtain the patterns we are looking for.
- **Output layer**: outputs the generated predictions.

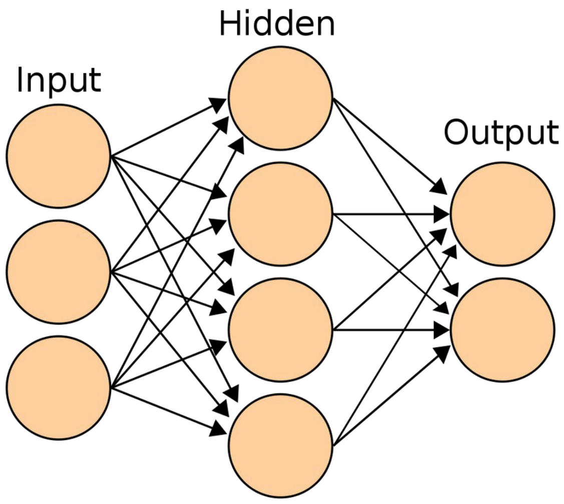


Figure 14. Neural network composition [54].

As stated above, this project is based on deep learning architectures, in it, we will apply the two most known categories of neural networks used in NLP tasks: Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Inside each kind of network we are able to create variations and different kinds of architectures, but the RNN networks go one step further, and contain variations that take different approaches to focus on different aspects or change certain mechanics in order to obtain better results depending on what we are looking for.

Besides the network architecture, deep learning neural networks contain a series of parameters that must be stated when designing the network. These parameters comprehend, besides others that can be designed by default:

- **Loss function**: function that will calculate the error between the expected or real output and the predicted output.
- **Optimizer function**: function that updates the weight parameters that the neurons contain in order to minimize the loss.
- **Number of epochs**: number of forward and backward pass of all the input samples for the training of the weights.
- **Batch size**: subset of samples that are trained at the same time, dividing each epoch into a number of iterations.

In this section we will have a brief introduction to each architecture, we are mostly interested in the results of the first two architectures, while the last two will be used for testing. There will be the code implementation that has been used in the project for each architecture. We will also take a look into to the different approaches that the papers have taken in the task along the years.

5.1. Convolutional Neural Network (CNN)

Convolutional neural networks (CNN) are a kind of neural network used in the late nineties that had gained relevance lately due to its good results on image recognition tasks [53]. They are based on the idea of identifying traits in the data in order to classify each sample. To do so, they use neurons specialized in two kind of operations: convolution and pooling. The first will have a filter of certain size, which will be a specified parameter, that will take information and reduce the total number of parameters in order to find characteristics that define the input. The second will support the first and group the data in order to reduce the input size, for it it is usually applied max-pooling, which takes the highest value in the filter used.

The architecture of a CNN at the hidden layers is composed of various convolutional layers followed each by their own max-pooling operations, once these layers are over, a fully-connected layer will process the data using a specific activation function to finally classify them applying a normalizing function.

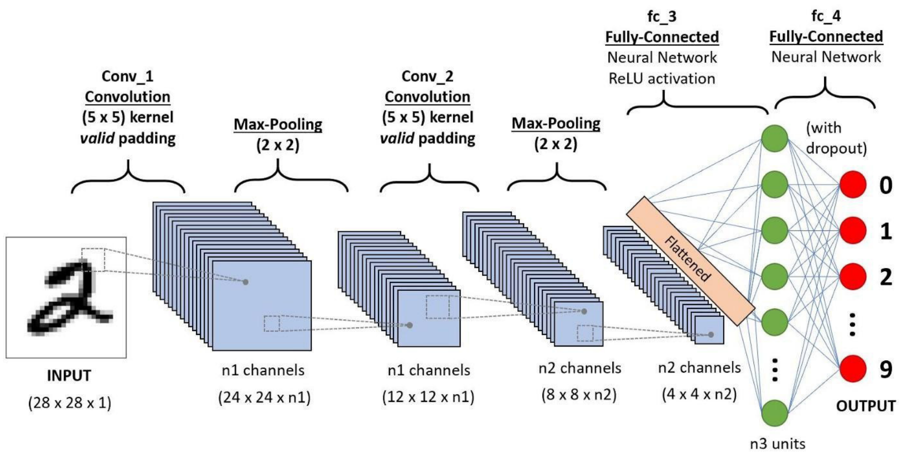


Figure 15. CNN sequence to classify handwritten digits [55].

If we take a look into the papers that use CNN for automatic sarcasm detection tasks, for example Mishra and Bhattacharyya [12] implement a specific kind of network with two different architecture complements that get merged at the end, but their starting points are equal, having one with a 1-dimensional convolution (text samples) and the other with a 2-dimensional (gaze samples) both followed each by their own max-pooling layer, results of both will get merged at the following layer to be feed to a fully-connected with dropout that will end up with a softmax for classification. The text component is the one we are interested in, it does a convolution of each sample to lately be max-pooled on the feature map. At Poria et al. [14] paper, we again find a neural network with more than one component, 3 in this case. One for emotion, one for sentiment and the last for personality. The layers are the same for each component, with a series of two convolution/max-pooling combinations, the first convolution has a filter of size 4 while at the second it is 3, both max-pooling are of size 2. After the three component parts, the three outputs from each are merged and go through a fully-connected layer again with dropout and end up with a softmax classifier. In this case the activation function is specified, which is a rectified linear unit (ReLu). The last network analized is a CNN used inside a model composed of different networks, including a LSTM, at Ghosh and Veale [15]. The CNN part is used in order to reduce frequency variation and map input features into composite robust features to be used as input for the LSTM. For it they use a convolution layer and its output is directly classified using a softmax layer which before going to the LSTM passes through a dropout. They discarded the usage of a max-pooling layer because keeping all the features for the LSTM improved their results.

The implementation used in this project will follow the same structure that we have seen along the different papers and theory concepts, by applying three rounds of convolution layers followed each by a max-pooling. A flatten layer is needed in order to transform the output from these layers before applying the densely connected layer with its activation function followed by the classifying layer using softmax.

The filters used are of size 2 for each layer, including the size of the max-pooling. The final activation layer will output 2 values since we are classifying between sarcastic and non sarcastic with a value for each. Every convolutional layer uses ReLu as its activation function. A summary of the network is available to figure out the different shapes and number of parameters that each layer of the network will have.

Layer (type)	Output Shape	Number of Parameters
1D Convolution	(None, 201, 16)	6,416
1D Max Pooling	(None, 100, 32)	0
1D Convolution	(None, 100, 16)	1,056
1D Max Pooling	(None, 50, 32)	0
1D Convolution	(None, 50, 16)	1,040
1D Max Pooling	(None, 25, 16)	0
Flatten	(None, 400)	0
Dense	(None, 16)	6416
Dense	(None, 2)	34

Total parameters	14,962
Trainable parameters:	14,962
Non-trainable parameters:	0

Table 13. Summary of the CNN network used.

5.2. Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNN) are a kind of neural networks specialized in analyzing temporal series of data in order to address the "time" dimension [53]. To do so, they have special kinds of neurons which "look back in time", getting feedback from neurons from the same layer. This way, each neuron remembers things learned from prior inputs while generating outputs, where parameters are shared between inputs, meaning that the length of the inputs will have to be the same for every input.

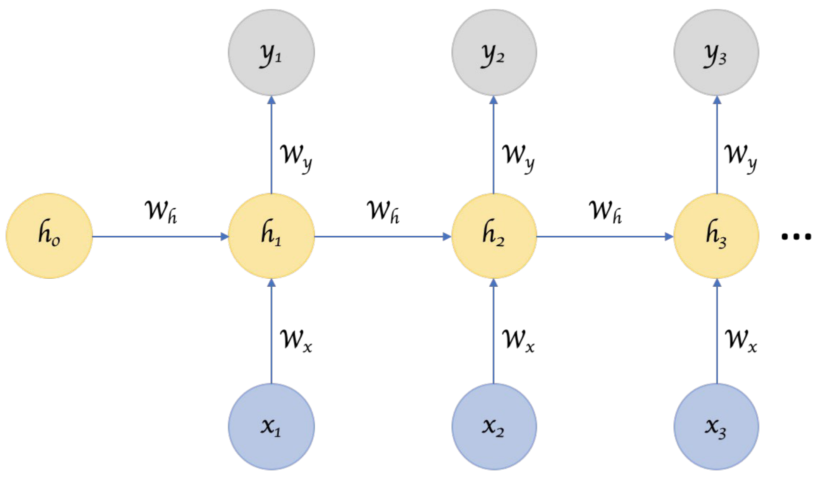


Figure 16. RNN detail, with a hidden state that is meant to carry pertinent information from one input item in the series to others [56].

In this project, three different kinds of RNN architectures are applied that have been utilized before at automatic sarcasm detection papers. These are Long Short-Term Memory (LSTM) networks, Bidirectional LSTM and Gated Recurrent Unit (GRU) networks. Various papers have applied LSTM architectures as their main network, while the last two have been complementary and will be used in order to see their results and to test various models to learn the way they can be implemented.

5.2.1. Long Short-Term Memory Network (LSTM)

Although RNN has context and gets feedback from the rest of the input, it is entirely possible for the gap between the relevant information and the point where it is needed to become very large [57]. That is the case when large data is used as input and here is where Long Short-Term Memory (LSTM) networks come to scene, which are specially designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior [57].

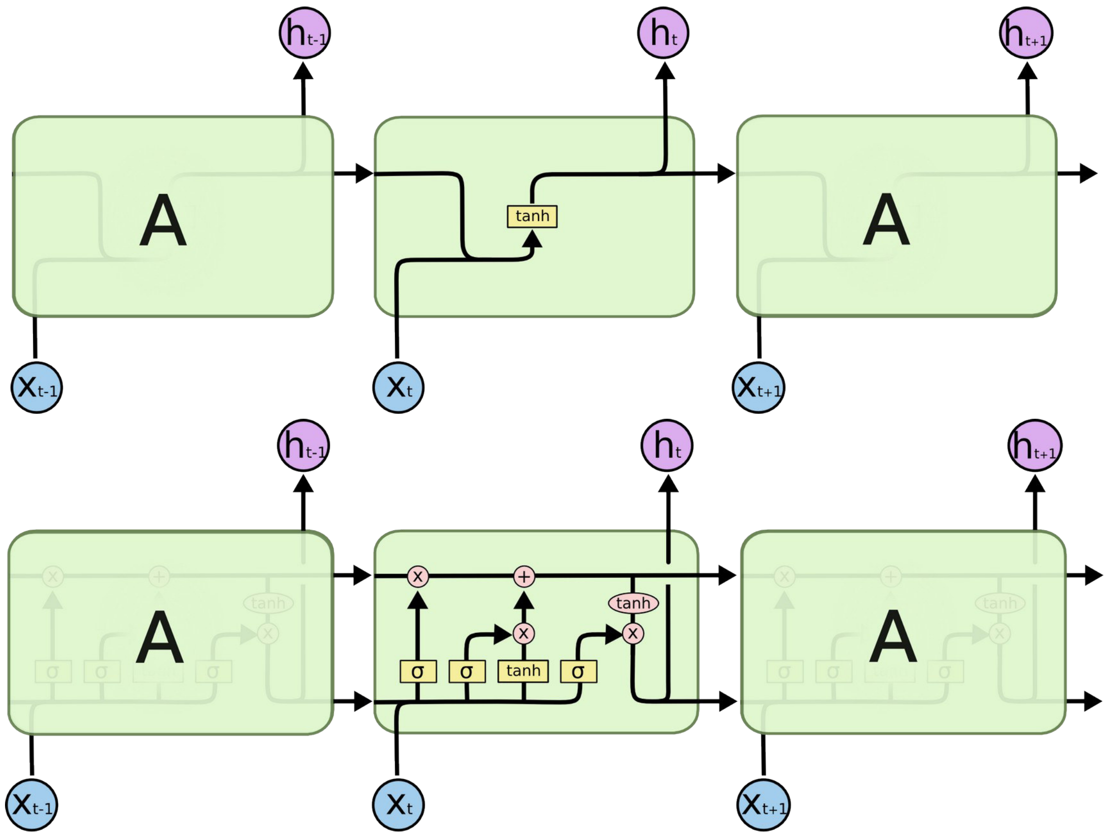


Figure 17. Comparison of the repeating modules from a RNN (top) and a LSTM (bottom) [57].

Taking a look at the papers that implement LSTM networks for automatic sarcasm detection, the change is usually in the number of LSTM layers and number of cells that they contain. At Kulkarni and Biswas [38] the layer is preceded by a spatial dropout, and succeeded by a dropout, although no information on the number of cells is mentioned, we can deduce that only one LSTM layer was applied. At Porwal et al. [58] two LSTM layers are used with 256 cells each, but there are no specifications on the other layers or processes taken. The last network analyzed at Ghosh and Veale [15] used LSTM along with a CNN, the CNN is used to map the features for the LSTM that will have 256 cells and a dropout, it will use sigmoid for classification.

The implementation used in this project will follow a similar structure to the ones we have seen along the different papers, but the LSTM layer will be slightly adapted reducing the number of cells to 128 since the tests were unsuccessful when the amount was 256, additionally the number of LSTM layers will be set to one. It will have a dense layer in order to pre-process the data for the LSTM layer, which will be the following with 128 cells as mentioned before, it will end with the classifying layer using softmax.

The final activation layer will output 2 values since we are classifying between sarcastic and non sarcastic with a value for each. A summary of the network is available to figure out the different shapes and number of parameters that each layer of the network will have.

Layer (type)	Output Shape	Number of Parameters
Dense	(None, 201, 64)	19,264
LSTM	(None, 100, 128)	98,816
Flatten	(None, 25728)	0
Dense	(None, 2)	51,458

Total parameters	169,538
Trainable parameters:	169,538
Non-trainable parameters:	0

Table 14. Summary of the LSTM network used.

5.2.2. Bidirectional LSTM

Inside the RNN architectures we can find a special kind called Bidirectional RNN that focus on getting additional context from the future in order to predict the past. In some tasks we need to know what's coming next to understand the context better and detect the present [56]. This is translated in automatic sarcasm detection to considering the entire sentence at once. Inside the Bidirectional RNN structure we will apply an LSTM architecture, leading to a Bidirectional LSTM network.

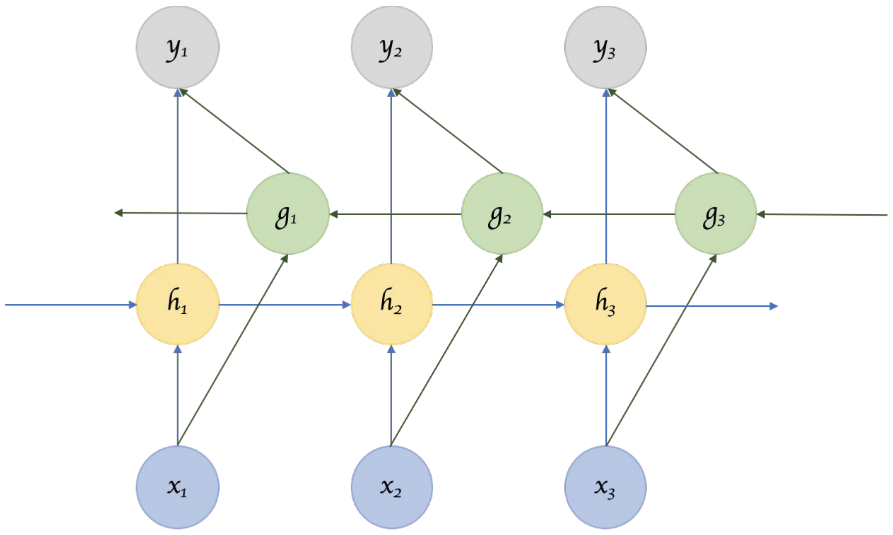


Figure 18. Bidirectional RNN structure [56].

This kind of network is applied at some papers, for example at Kumar, Sangwan, Arora, Nayyar and Abdel-Basset [39] a Bidirectional LSTM is used in a custom network as a layer along with other different layers such as a convolution layer. Its purpose is to to learn high-level features from the previous layer. It appears interesting to test its efficiency alone in a non-custom network.

The implementation used in this project will be a basic network following the simple but efficient models of structures that the project has shown before for the CNN and LSTM networks. It will contain two bidirectional layers using LSTM for both forward and backward layers. It will end with the densely connected layer for classifying using softmax.

The LSTM that will compose the Bidirectional network will have 64 and 32 cells each. The final activation layer will output 2 values since we are classifying between sarcastic and non sarcastic with a value for each. A summary of the network is available to figure out the different shapes and number of parameters that each layer of the network will have.

Layer (type)	Output Shape	Number of Parameters
Bidirectional	(None, 201, 128)	135,680
Bidirectional	(None, 64)	41,216
Dense	(None, 2)	130

Total parameters	177,026
Trainable parameters:	177,026
Non-trainable parameters:	0

Table 15. Summary of the Bidirectional LSTM network used.

5.2.3. Gate Recurrent Unit Network (GRU)

Introduced by Cho et al. in 2014, the Gate Recurrent Unit network aims to solve a problem RNN has, the vanishing gradient problem, which prevents the network's weights from updating correctly. GRU can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results [60]. In order to save that problem, they introduce an update gate and reset gate, which will manage the information that the output will obtain. The update gate manages the information from the previous time steps that will get passed along to the following, while the reset gate decides how much information from the previous time steps will be forgotten. The aim is to keep the relevant information along the different time steps of the network.

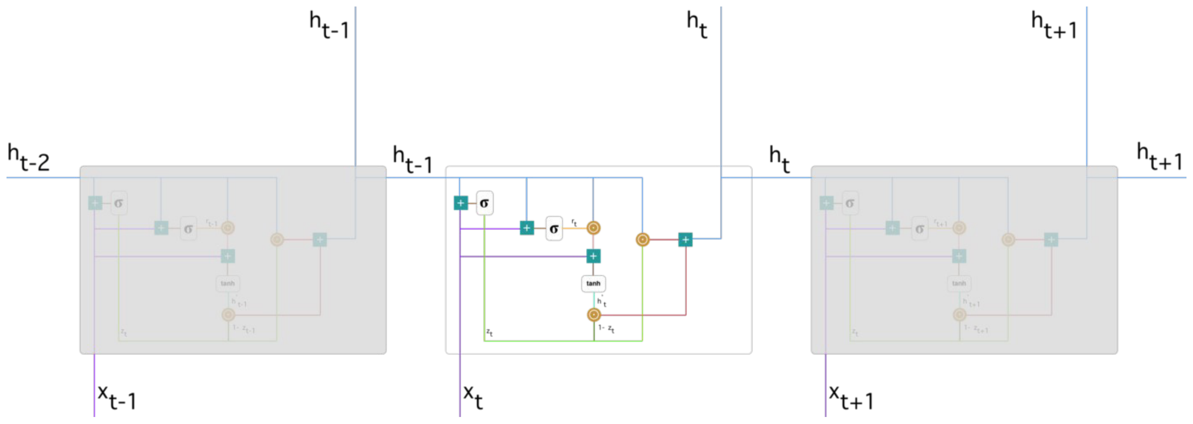


Figure 19. GRU structure [60].

This kind of network has been applied at some points, like at Majumder, Poria, Peng, Chhaya, Cambria and Gelbukh [13], where a GRU is used for sentence representation, but they mention in their experiments section that they also tried applying it for their simple multitask classifier and two separated GRU for a shared attention model, although this application did not suppose an improvement compared to the results from the original model. Following the testing and learning from the project, it looks interesting to test that kind of network and its results for this project.

The implementation of this network for this project will mimic the LSTM implementation, changing the LSTM layer for a GRU one. Since it is not possible for automatic sarcasm detection task to find papers using a GRU as their main network alone or detailing the layers used, and given the fact that GRU are a variation of the LSTM, it appears as the most fitting way to proceed. This means that it will have a dense layer in order to preprocess the data for the GRU layer, which will be the following, it will end with a classifying layer using softmax after the data has been flattened. The GRU layer will contain 128 cells in an attempt to follow the LSTM structure and parameters. The final activation layer will output 2 values since we are classifying between sarcastic and non sarcastic with a value for each. A summary of the network is available to figure out the different shapes and number of parameters that each layer of the network will have.

Layer (type)	Output Shape	Number of Parameters
1D Convolution	(None, 201, 64)	19,264
1D Max Pooling	(None, 201, 128)	74,496
1D Convolution	(None, 25728)	0
1D Max Pooling	(None, 2)	51,458

Total parameters	145,218
Trainable parameters:	145,218
Non-trainable parameters:	0

Table 16. Summary of the GRU network used.

6. Experiments

After taking a look at the different word embeddings and architectures that will be used in the tests, it is time to see the different tests and results that have been performed. Before proceeding we will present a series of objectives to find if they come as true or not.

1. The usage of a stemmer on the corpus will improve the results for models trained from scratch but it might not work properly with pre-trained word embeddings unless they contain stems in their vocabulary.
2. The best architecture globally should be the CNN, since it is the most used and better detailed along the papers about how the layers should be. The dataset has also been used before in CNN architectures, which should be the best fit for it. Since the architecture is not the same, the accuracy might not be as good as in the papers.
3. The LSTM architectures might show good performances, but probably not as good as CNN architectures.
4. The results for the last two architectures (Bidirectional LSTM and GRU) are uncertain since their purpose is to be tested in this field and there are no high expectations on them. They should give results similar to the LSTM architecture.
5. The word embeddings can either be trained at the moment or pre-trained, if we look at their usage and information published along different papers, the performance of the pre-trained models appear to be better and the most followed way. There is some uncertainty on the performance comparison of a big vocabulary vs a short, focused vocabulary.

For the structure to classify the tests we will separate each test based on the type of word embeddings they use, we have four groups that will compose this section:

- **Category 1**: Word2Vec
- **Category 2**: Word2Vec pre-trained
- **Category 3**: GloVe
- **Category 4**: GloVe pre-trained

Inside each group the tests might vary, considering that the word embeddings size will be 300 for Word2Vec category tests and 200 for GloVe, we will have have the following basic elements to be tested:

- Usage of stemming or not
- Number of epochs

Except for some of the Category 2 tests, every test will use Adam as their optimizer function and Categorical Cross Entropy as their loss function. This is the result of a series of tests with the diverse functions that the TensorFlow library contains. This is not strange since Categorical Cross Entropy is the most commonly used loss function [61], and Adam is one of the best optimizer functions for neural networks [62].

Besides the mentioned parameters or functions that will be used, there are specific cases based on the category:

- For Category 1: Word2Vec can be trained using two different models.
  - Word2Vec using Skip-gram
  - Word2Vec using Continuous Bag of Words
- For Category 2: two groups of optimizer and loss function had successful results for this category.
  - Adam optimizer with Categorical Cross Entropy loss functions
  - Nadam optimizer with Binary Cross Entropy loss functions

The parameters that are applied or change will be specified and pointed out at their corresponding test section. The discussion section (Section 6.3.) will contain the conclusions that can be taken from both Sections 6.1. and 6.2., where only the obtained results are displayed.

6.1. Tests

Each test in this section has been performed 10 times in order for those results to be averaged to reduce the randomness factor that each training contains. The results given are the outcome of this. The number of epochs applied in order to get the optimal results was sometimes different based on trial and error applied at every test, which ended up giving a number of epochs that showed good results compared to the rest. Epoch testing was carried out with values up to 100. Data management will be 80% for training and 20% for testing at every test.

At the tables, the different parameters used will be shown in square brackets. The legend is as follows:

Parameter	**Representation in square brackets**
Usage of stemming	S
Non usage of stemming	NS
Number of epochs	*number*
Usage of Skip-gram (Word2Vec)	SG
Usage of CBOW (Word2Vec)	CBOW

Table 17. Legend for the parameters applied in the tests.

6.1.1. Word2Vec.

In this first series of tests, the architectures are evaluated using Word2Vec word embeddings generated by training from scratch at the moment. As stated in Section 4.1.1, these word embeddings can be obtained either applying the Skip-gram or CBOW model. In order to test both configurations, each test will be repeated for both models. Text modeling applying stemming or not will be specified, repeating every test for both types.

Model	**Accuracy**	**Loss**
CNN [SG,10,S]	85.75%	44.69%
<u>CNN [SG,10,NS]</u>	**<u>87.06%</u>**	**<u>43.86%</u>**
CNN [CBOW,10,S]	85.70%	44.74%
CNN [CBOW,10,NS]	84.40%	46.00%
LSTM [SG,10,S]	84.90%	46.14%
LSTM [SG,10,NS]	**85.60%**	**45.29%**
LSTM [CBOW,10,S]	84.40%	46.09%
LSTM [CBOW,10,NS]	83.65%	47.12%
BLSTM [SG,15,S]	**86.92%**	**44.48%**
BLSTM [SG,15,NS]	85.85%	45.06%
BLSTM [CBOW,15,S]	84.20%	46.65%
BLSTM [CBOW,15,NS]	82.50%	48.04%
GRU [SG,20,S]	**86.51%**	**44.51%**
GRU [SG,20,NS]	85.65%	44.95%
GRU [CBOW,20,S]	85.95%	44.98%

Table 18. Test results for Word2Vec trained from scratch.

The test corresponding to GRU [CBOW,20,NS] was outputting low results and was skipped. The BLSTM [SG,15,S] test carried out the highest results overall in a test with a 92% of accuracy and 39,7% of loss in one of the 10 tests. The best result for each architecture is highlighted in bold. The best result overall is underlined.

Graphical representation for the training process and results of bold results are available at the appendix section.

6.1.2. Word2Vec pre-trained.

In this second series of tests, the architectures are evaluated using the pre-trained Word2Vec word embeddings previously introduced, which has been trained using the Skip-gram model [63]. In this case, applying stemming was ineffective and gave worse results compared to not applying it, for this reason the stemming tests were removed. In order to recognise the tests using Nadam optimizer with Binary Cross Entropy loss functions, they have been highlighted with an asterisk (*).

Model	**Accuracy**	**Loss**
CNN* [SG,20,NS]	88.82%	55.96%
CNN [SG,20,NS]	**87.86%**	**42.72%**
LSTM [SG,20,NS]	87.57%	43.80%
BLSTM* [SG,15,NS]	87.20%	56.69%
BLSTM [SG,15,NS]	85.51%	45.38%
GRU* [SG,20,NS]	88.22%	56.07%
GRU [SG,20,NS]	86.20%	44.59%

Table 19. Test results for pre-trained Word2Vec.

Although the different optimizer and loss functions worked here, their performance reported some problems, causing the model to fail to improve at some moments and overall giving a higher loss result, considering this, CNN* [20] achieved the highest accuracy in a test with 94% accuracy but giving a 54.2% loss. The performed tests failed more than succeeded for LSTM when applying the alternative functions. For the standard optimizer and loss functions, CNN [20] and LSTM [20] achieved results higher than 91% in some tests.

Graphical representation for the training process and results of the best test overall, which is highlighted in bold, is available at the appendix section.

6.1.3. GloVe.

In this third series of tests, the architectures are evaluated using GloVe word embeddings generated by training from scratch at the moment. As opposed to the previous tests, not applying stemming resulted in low results compared to its application, leading to the removal of the group of tests without stemming.

Model	Accuracy	Loss
CNN [15,S]	79.85%	50.82%
LSTM [15,S]	79.10%	51.44%
BLSTM [15,S]	83.25%	47.49%
GRU [15,S]	**83.45%**	**47.47%**

Table 20. Test results for GloVe trained from scratch.

Graphical representation for the training process and results of the best test overall, which is highlighted in bold, is available at the appendix section.

6.1.4. GloVe pre-trained.

For the fourth and last series of tests, the architectures are evaluated using the pre-trained GloVe word embeddings previously introduced. As we saw before in the previous Word2Vec pre-trained tests (Section 6.1.2.), applying stemming was ineffective and gave worse results compared to not applying it, for this reason the stemming tests were removed again for this series of tests.

Model	Accuracy	Loss
CNN [15,NS]	84.91%	46.08%
LSTM [30,NS]	84.00%	46.60%
BLSTM [15,NS]	**85.20%**	**45.49%**
GRU [20,NS]	84.65%	45.83%

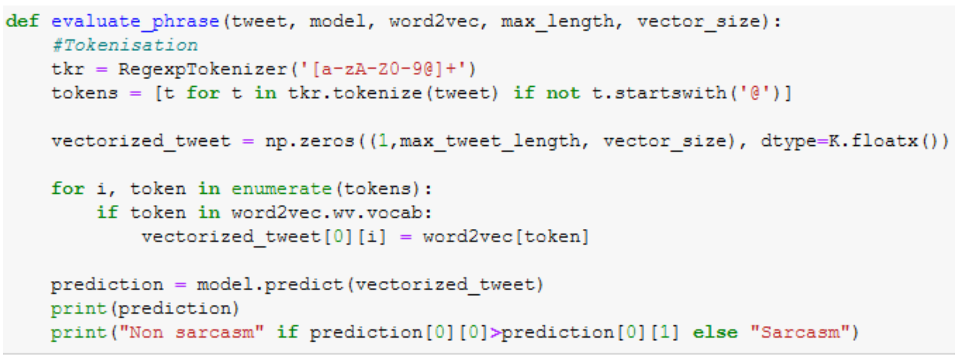
Table 21. Test results for pre-trained GloVe.

Graphical representation for the training process and results of the best test overall, which is highlighted in bold, is available at the appendix section.

6.2. Predictions

For this section, the two best models overall will be chosen in order to test phrases and their effectivity. The chosen models are for Word2Vec CNN [SG,10,NS], with the second best result overall, and for pre-trained Word2Vec CNN [SG,20,NS], with the best result overall.

The evaluation takes place using a developed function that will emulate the process taken for word embedding the dataset doing the same for a single phrase, then using the predict function that Word2Vec has it will display its results.



```
def evaluate_phrase(tweet, model, word2vec, max_length, vector_size):
    #Tokenisation
    tkr = RegexpTokenizer('[a-zA-Z0-9@]+')
    tokens = [t for t in tkr.tokenize(tweet) if not t.startswith('@')]

    vectorized_tweet = np.zeros((1,max_tweet_length, vector_size), dtype=K.floatx())

    for i, token in enumerate(tokens):
        if token in word2vec.wv.vocab:
            vectorized_tweet[0][i] = word2vec[token]

    prediction = model.predict(vectorized_tweet)
    print(prediction)
    print("Non sarcasm" if prediction[0][0]>prediction[0][1] else "Sarcasm")
```

Figure 20. Code for evaluating phrases with the trained model, which has a slight modification for pre-trained.

In this section, both models are evaluated having each a section, where phrases from dataset 2 and 3 are tested, as it happened with the tests before, the conclusions that can be taken from its results will be located at the discussion section (Section 6.3.).

6.2.1. CNN with Word2Vec

Correctly guessed answers are colored as green, while wrong will be red. Firstly we will have the table of results for dataset 2, with the table of results for dataset 3 shown afterwards.

Text	**Prediction**
Tell you what-get your math and science skills up to a grade 6 level and I'll consider taking advice from you, 'kay?	Sarcasm
That must be why they like to wear boots in Texas.	Sarcasm
In England we have a sack with a couple of bricks inside it.	Not sarcasm
And you wonder why people call "republicans" racist, women hating bigots. (hint: it's comments like yours)	Sarcasm
Reading Difficulties, huh? There's a cure for that. It's called school. Perhaps you should go there.....	Sarcasm
Marriage is a secular condition set up by various Societies over the millenia and one of the prerequisites set out is that such a union should be between people of different genders.Religion has supported this and sanctified it but did not invent it.You don't curse the Church you lobby Society to see if you can effect a change. People who believe the different gender aspect are entitled to lobby for no change.	Not sarcasm
where did you get this from? i want to read it.	Sarcasm
i love how you claim that the teabag party is not affiliated with the republicans, who are they going to vote with ?.. the democrats?.. they may not	Not sarcasm

claim to be republicans but last i checked we only have two major parties representing us in the house and senate, with maybe one or two independents.the tea bag party isn't big enough to control anything without republicans so enlighten me, who are they going to vote with to destroy what obama has accomplished so far?...	
i just love how deliciously ironic this statement here is. well done.	Not sarcasm
reckon if a rock could think it would have a concept or two.	Not sarcasm
i love how every line, except number 3, has a winking emoticon at the end. haha	Not sarcasm

Table 22.CNN with trained at the moment Word2Vec predictions for Dataset 2.

Text	**Prediction**
You do know west teams play against west teams more than east teams right?	Sarcasm
gotta love the teachers who give exams on the day after halloween	Sarcasm
Let's just make everyone poor so they don't drive.	Sarcasm
Yep, people who use critical-thinking skills professionally are obviously biased irrationally.	Sarcasm
But "it says camera on top of it?!"	Not sarcasm
Good advertisement for Andre's steam group though...	Not sarcasm
Get a tattoo on my forehead saying I told you so.	Sarcasm
And they say romance is dead...	Not sarcasm
I was really surprised when it did not work.	Sarcasm
pretty sure, given it was 12+ years ago, its long past any of that.	Not sarcasm
OH that's a reliable news site.	Sarcasm
Quality over quantity had never been more appropriate	Sarcasm

Table 23.CNN with trained at the moment Word2Vec predictions for Dataset 3.

6.2.2. CNN with pre-trained Word2Vec

Correctly guessed answers are colored as green, while wrong will be red. Firstly we will have the table of results for dataset 2, with the table of results for dataset 3 shown afterwards.

Text	Prediction
Tell you what-get your math and science skills up to a grade 6 level and I'll consider taking advice from you, 'kay?	Sarcasm
That must be why they like to wear boots in Texas.	Sarcasm
In England we have a sack with a couple of bricks inside it.	Sarcasm
And you wonder why people call "republicans" racist, women hating bigots. (hint: it's comments like yours)	Sarcasm
Reading Difficulties, huh? There's a cure for that. It's called school. Perhaps you should go there.....	Not sarcasm
Marriage is a secular condition set up by various Societies over the millenia and one of the prerequisites set out is that such a union should be between people of different genders.Religion has supported this and sanctified it but did not invent it.You don't curse the Church you lobby Society to see if you can effect a change. People who believe the different gender aspect are entitled to lobby for no change.	Sarcasm
where did you get this from? i want to read it.	Not sarcasm
i love how you claim that the teabag party is not affiliated with the republicans, who are they going to vote with ?.. the democrats?.. they may not claim to be republicans but last i checked we only have two major parties representing us in the house and senate, with maybe one or two independents.the tea bag party isn't big enough to control anything without republicans so enlighten me, who are they going to vote with to destroy what obama has accomplished so far?...	Not sarcasm
i just love how deliciously ironic this statement here is. well done.	Sarcasm
reckon if a rock could think it would have a concept or two.	Sarcasm
i love how every line, except number 3, has a winking emoticon at the end. haha	Sarcasm

Table 24. CNN with pre-trained Word2Vec predictions for Dataset 2.

Text	**Prediction**
You do know west teams play against west teams more than east teams right?	Sarcasm
gotta love the teachers who give exams on the day after halloween	Sarcasm
Let's just make everyone poor so they don't drive.	Sarcasm
Yep, people who use critical-thinking skills professionally are obviously biased irrationally.	Not sarcasm
But "it says camera on top of it?!"	Sarcasm
Good advertisement for Andre's steam group though...	Not sarcasm
Get a tattoo on my forehead saying I told you so.	Sarcasm
And they say romance is dead...	Sarcasm
I was really surprised when it did not work.	Sarcasm
pretty sure, given it was 12+ years ago, its long past any of that.	Not sarcasm
OH that's a reliable news site.	Sarcasm
Quality over quantity had never been more appropriate	Not sarcasm

Table 25. CNN with pre-trained Word2Vec predictions for Dataset 3.

6.3. Discussion

The aim of the different tests was firstly to try to achieve a result similar to the work from Mishra et al. [12] regarding the CNN architecture, but also to test the other kinds of architectures and see their performance results. The best accuracy overall was reached at the Word2Vec pre-trained test CNN* [SG,20,NS], with an 88.82% value, this test was performed on a different set of optimizer and loss functions, Nadam and Binary Cross Entropy respectively, than the used for the majority of tests. Those functions report an increase of more than a 10% regarding its loss value compared to the following best result with a different set of functions for the same architecture and word embeddings but with Adam and Categorical Cross Entropy, with 87.86% accuracy, Word2Vec pre-trained test CNN [SG,20,NS]. The increase of loss could mean that there is overfitting, and since the difference is large, we will consider as the best result the last. This result is almost the same that the best result obtained at Mishra et al. [12], 87.42%, considering the randomness and various factors that could have influenced and that there is also no specific clarification on what kind of word embeddings they used. It is also interesting the fact that the second best result is also from a CNN configuration, but this time from the Word2Vec group, CNN [SG,10,NS] which achieves an 87,06% of accuracy, with a slightly higher loss value than the first best result.

As it could be seen, Skip-gram is the most accurate model for Word2Vec overall, which confirms the statement from Mikolov et al. [63] mentioned before, that Skip-gram is slower but better for infrequent words, while CBOW is simply faster. Although there is no pre-trained CBOW model, there was no improvement at the trained from scratch Word2Vec word embeddings when switching from Skip-gram to CBOW.

Another aim that the testing had was to see how the architectures different than CNN performed, which performed considerably good considering their architecture compositions were fairly simple. With a special mention to the Word2Vec pre-trained test LSTM [SG,20,NS] which got an accuracy of 87.57% and the Word2Vec tests BLSTM [SG,15,S] and GRU [SG,20,S] with accuracies of 86.92% and 86.51% respectively.

It is relevant that for Word2Vec trained at the moment, CNN and LSTM obtained their best results without stemming, while BLSTM and GRU needed stemming. There is also a decrease in accuracy when no stemming is applied for the CBOW tests. As the best results came without stemming, although there is no clear pattern to recommend the usage or not of it, one could say that it is not needed when the right tools are applied, but it could come useful if our approach is different, statement that would need to be proved with real world prediction tests.

When it comes to the different word embeddings used, GloVe did not perform as good as Word2Vec, there is also a huge gap when comparing every category of word embeddings tests with the GloVe trained at the moment tests, which not only gave worse results overall but they only worked properly with stemming. Although GloVe pre-trained word embeddings tests had performances with good numbers, they did not reach the values that Word2Vec achieved. It could be interesting to test their performance on real case scenarios.

In addition, the predictions section was aimed to test the performance of the two best models with cases that were different than the dataset samples. The obtained results are very interesting and, although they are not a valid representation of their accuracy in a “real world” environment, they serve as a first approach to it, where we can see how the pre-trained model performs slightly better, where some wrong predictions can be correctly classified at the model using standard Word2Vec.

It is palpable how large texts are usually wrongly classified, where they are frequently classified as not sarcasm. This is better understood when we take into account that the majority of the texts from the training dataset were short, due to the fact that Twitter has a 280 character limit, meaning the models that are trained for Twitter should only be used for the corresponding cases. When we take a look at the datasets, it is clear how Dataset 2 is more informal compared to Dataset 3, leading to more words out of vocabulary. Also since Dataset 2 is composed of replies, their messages are hardly understood out of context, which is interestingly reflected to the predictions, where they are hardly correctly classified. If the phrases can be understood out of context, it can be seen how short and average sized texts are more than usually correctly classified, which are some nice interesting results overall and confirm the good performance of the models.

7. Conclusions

At first when I started this project I did barely know how to build a neural network, just some concepts and a basic way to do so with the Pytorch library. I had no experience on how to address a real project or how to use deep neural networks for a NLP task. Throughout the course of finding information and learning how to address this project, I learned about the TensorFlow library and its newest version which made the task much easier, I also learned how to address a deep learning NLP task since I only knew how to use it for images and I only had theoretical and surface level knowledge on word embeddings.

It was very interesting to build the whole project, where finding an adequate dataset took some time, but luckily Dataset 1 came very handy with quality texts and annotations. The lookup was not as successful for the Spanish language, since at first the project was meant to test both languages. The predictions section came late in the project, which was a fun section to do and execute, and gives a practical functionality to test the models in real life situations.

Doing the state of the art was very instructive, and made me realize that it is not normal to include technical details at papers, which makes it difficult trying to replicate their work. There have been various interesting approaches lately for automatic sarcasm detection, where assembling of different architectures appear to be giving good results. It would have been interesting to test a hybrid approach like that, but finding the best approach and testing it would have changed the whole project, where it felt like testing a group of basic approaches was the most adequate way to address it, which also gave good results as it did at the papers this was carried out. For the interested reader, the paper by Joshi et al. [11] offers a comprehensive review of the state of the art, with an in-depth analysis of the datasets that have been used in the papers that are analyzed there.

There have been some complications along the project, which have been solved successfully. At first before starting the code section, it was hard to find a way to create the project code, and libraries like Theano or Pytorch were tested before trying TensorFlow. Then the testing, which at first was carried out with Dataset 2, was giving bad results due to its particular composition, where it could be interesting to try an approach with two components that can analyze quote-response combinations, which could give better results when detecting sarcasm in those situations. Finally, finding papers with detailed or basic architectures for Bidirectional LSTM and GRU was not possible, a fact that changed the way the project was being faced, being then centered around CNN and LSTM with the other two mentioned before as test architectures to see their performances.

For future work or improvement of this project, hybrid architectures could be tested, and the dataset could be created from scratch, which is not an easy task as seen along the datasets used in the different state of the art papers.

8. References

[1] Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, *2*(1–2), 1-135.

[2] Nigam, K., & Hurst, M. (2006). Towards a robust metric of polarity. In *Computing Attitude and Affect in Text: Theory and Applications* (pp. 265-279). Springer, Dordrecht.

[3] Davidov, D., Tsur, O., & Rappoport, A. (2010, July). Semi-supervised recognition of sarcastic sentences in twitter and amazon. In Proceedings of the fourteenth conference on computational natural language learning (pp. 107-116). Association for Computational Linguistics.

[4] González-Ibáñez, R., Muresan, S., & Wacholder, N. (2011, June). Identifying sarcasm in Twitter: a closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers-Volume 2* (pp. 581-586). Association for Computational Linguistics.

[5] Reyes, A., Rosso, P., & Buscaldi, D. (2012). From humor recognition to irony detection: The figurative language of social media. *Data & Knowledge Engineering*, *74*, 1-12.

[6] Riloff, E., Qadir, A., Surve, P., De Silva, L., Gilbert, N., & Huang, R. (2013, October). Sarcasm as contrast between a positive sentiment and negative situation. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (pp. 704-714).

[7] Maynard, D. G., & Greenwood, M. A. (2014, March). Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. In *LREC 2014 Proceedings*. ELRA.

[8] Rajadesingan, A., Zafarani, R., & Liu, H. (2015, February). Sarcasm detection on twitter: A behavioral modeling approach. In *Proceedings of the eighth ACM international conference on web search and data mining* (pp. 97-106).

[9] Bamman, D., & Smith, N. A. (2015, April). Contextualized sarcasm detection on twitter. In *Ninth International AAAI Conference on Web and Social Media*.

[10] Joshi, A., Sharma, V., & Bhattacharyya, P. (2015, July). Harnessing context incongruity for sarcasm detection. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)* (pp. 757-762).

[11] Joshi, A., Bhattacharyya, P., & Carman, M. J. (2017). Automatic sarcasm detection: A survey. *ACM Computing Surveys (CSUR)*, *50*(5), 1-22.

[12] Mishra, A., Dey, K., & Bhattacharyya, P. (2017, July). Learning cognitive features from gaze data for sentiment and sarcasm classification using convolutional neural network. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 377-387).

[13] Majumder, N., Poria, S., Peng, H., Chhaya, N., Cambria, E., & Gelbukh, A. (2019). Sentiment and sarcasm classification with multitask learning. *IEEE Intelligent Systems*, *34*(3), 38-43.

[14] Poria, S., Cambria, E., Hazarika, D., & Vij, P. (2016). A deeper look into sarcastic tweets using deep convolutional neural networks. arXiv preprint arXiv:1610.08815.

[15] Ghosh, A., & Veale, T. (2016, June). Fracking sarcasm using neural network. In Proceedings of the 7th workshop on computational approaches to subjectivity, sentiment and social media analysis (pp. 161-169).

[16] Camp, E. (2012). Sarcasm, pretense, and the semantics/pragmatics distinction. Noûs, 46(4), 587-634.

[17] Tepperman, J., Traum, D., & Narayanan, S. (2006). " Yeah Right": Sarcasm Recognition for Spoken Dialogue Systems. In Ninth international conference on spoken language processing.

[18] Abercrombie, G., & Hovy, D. (2016, August). Putting sarcasm detection into context: The effects of class imbalance and manual labelling on supervised machine classification of twitter conversations. In Proceedings of the ACL 2016 Student Research Workshop (pp. 107-113).

[19] Mishra, A., Kanojia, D., & Bhattacharyya, P. (2016, March). Predicting readers' sarcasm understandability by modeling gaze behavior. In Thirtieth AAAI conference on artificial intelligence.

[20] Joshi, A., Mishra, A., Senthamilselvan, N., & Bhattacharyya, P. (2014, June). Measuring sentiment annotation complexity of text. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (pp. 36-41).

[21] Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

[22] Joshi, A., Sharma, V., & Bhattacharyya, P. (2015, July). Harnessing context incongruity for sarcasm detection. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers) (pp. 757-762).

[23] Mishra, A., Kanojia, D., Nagar, S., Dey, K., & Bhattacharyya, P. (2017). Leveraging cognitive features for sentiment analysis. arXiv preprint arXiv:1701.05581.

[24] Wallace, B. C., Kertz, L., & Charniak, E. (2014, June). Humans require context to infer ironic intent (so computers probably do, too). In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (pp. 512-516).

[25] Preotiuc-Pietro, D., Samangooei, S., Cohn, T., Gibbins, N., & Niranjan, M. (2012, May). Trendminer: An architecture for real time analysis of social media text. In Sixth International AAAI Conference on Weblogs and Social Media.

[26] Carvalho, P., Sarmento, L., Silva, M. J., & De Oliveira, E. (2009, November). Clues for detecting irony in user-generated contents: oh...!! it's" so easy";-. In Proceedings of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion (pp. 53-56).

[27] Ramteke, A., Malu, A., Bhattacharyya, P., & Nath, J. S. (2013, August). Detecting turnarounds in sentiment analysis: Thwarting. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (pp. 860-865).

[28] Kianna. (2016, June 8). Difference Between Irony and Sarcasm. Retrieved April 7, 2020, from http://www.differencebetween.net/miscellaneous/difference-between-irony-and-sarcasm/

[29] Camp, E. (2012). Sarcasm, pretense, and the semantics/pragmatics distinction. Noûs, 46(4), 587-634.

[30] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[31] Barbieri, F., Saggion, H., & Ronzano, F. (2014, June). Modelling sarcasm in twitter, a novel approach. In Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis (pp. 50-58).

[32] Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., & Lai, J. C. (1992). Class-based n-gram models of natural language. Computational linguistics, 18(4), 467-479.

[33] Owoputi, O., O’Connor, B., Dyer, C., Gimpel, K., Schneider, N., & Smith, N. A. (2013, June). Improved part-of-speech tagging for online conversational text with word clusters. In Proceedings of the 2013 conference of the North American chapter of the association for computational linguistics: human language technologies (pp. 380-390).

[34] Sarcasm. (n.d.). Retrieved May 6, 2020, from https://www.thefreedictionary.com/sarcasm

[35] Source code for nltk.stem.lancaster. (2020, April 13). Retrieved May 11, 2020, from https://www.nltk.org/_modules/nltk/stem/lancaster.html

[36] A Beginner's Guide to Word2Vec and Neural Word Embeddings. (n.d.). Retrieved May 13, 2020, from https://pathmind.com/wiki/word2vec

[37] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

[38] Kulkarni, S., & Biswas, A. (2020). Sarcasm Detection Methods in Deep Learning: Literature Review. In *ICT Analysis and Applications* (pp. 507-512). Springer, Singapore.

[39] Kumar, A., Sangwan, S. R., Arora, A., Nayyar, A., & Abdel-Basset, M. (2019). Sarcasm detection using soft attention-based bidirectional long short-term memory model with convolution network. *IEEE Access*, *7*, 23319-23328.

[40] Pang, B., & Lee, L. (2004, July). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics* (p. 271). Association for Computational Linguistics.

[41] Oraby, S., Harrison, V., Reed, L., Hernandez, E., Riloff, E., & Walker, M. (2017). Creating and characterizing a diverse corpus of sarcasm in dialogue. *arXiv preprint arXiv:1709.05404*.

[42] Walker, M. A., Tree, J. E. F., Anand, P., Abbott, R., & King, J. (2012, May). A Corpus for Research on Deliberation and Debate. In *LREC* (Vol. 12, pp. 812-817).

[43] Riloff, E. (1996, August). Automatically generating extraction patterns from untagged text. In *Proceedings of the national conference on artificial intelligence* (pp. 1044-1049).

[44] akshit3050. (2018, August 04). Sarcasm Dataset. Retrieved May 19, 2020, from https://www.kaggle.com/akshit3050/sarcasm/data

[45] English Gigaword Fifth Edition. (2011, June 17). Retrieved May 20, 2020, from https://catalog.ldc.upenn.edu/LDC2011T07

[46] 3Top. (2017, November 3). 3Top/word2vec-api. Retrieved May 22, 2020, from https://github.com/3Top/word2vec-api

[47] Barazza, L. (2017, April 3). How does Word2Vec's Skip-Gram work? Retrieved May 22, 2020, from https://becominghuman.ai/how-does-word2vecs-skip-gram-work-f92e0525def4

[48] NLP with gensim (word2vec). (n.d.). Retrieved May 22, 2020, from https://www.samyzaf.com/ML/nlp/nlp.html

[49] Hui, J. (2019, November 5). NLP - Word Embedding & GloVe. Retrieved May 23, 2020, from https://medium.com/@jonathan_hui/nlp-word-embedding-glove-5e7f523999f6

[50] Perone, C. S., Silveira, R., & Paula, T. S. (2018). Evaluation of sentence embeddings in downstream and linguistic probing tasks. *arXiv preprint arXiv:1806.06259*.

[51] Pennington, J. (n.d.). Retrieved May 24, 2020, from
https://nlp.stanford.edu/projects/glove/

[52] Simon, G. (2017, March 30). GradySimon/tensorflow-glove. Retrieved May 24, 2020, from https://github.com/GradySimon/tensorflow-glove

[53] Torres, J. (2020). *Python deep learning introducción práctica con Keras y TensorFlow 2* (1st ed.). Barcelona: Marcombo.

[54] Artificial Neural Networks/Neural Network Basics. (n.d.). Retrieved May 25, 2020, from https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Neural_Network_Basics

[55] Saha, S. (2018, December 17). A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. Retrieved May 25, 2020, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[56] Venkatachalam, M. (2019, June 22). Recurrent Neural Networks. Retrieved May 26, 2020, from https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

[57] Olah, C. (2015, August 27). Understanding LSTM Networks. Retrieved May 26, 2020, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[58] Porwal, S., Ostwal, G., Phadtare, A., Pandey, M., & Marathe, M. V. (2018, June). Sarcasm Detection Using Recurrent Neural Network. In *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)* (pp. 746-748). IEEE.

[59] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[60] Kostadinov, S. (2019, November 10). Understanding GRU Networks. Retrieved May 29, 2020, from https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[61] Parmar, R. (2018, September 2). Common Loss functions in machine learning. Retrieved June 2, 2020, from https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

[62] Peixeiro, M. (2020, May 19). The 3 Best Optimization Methods in Neural Networks. Retrieved June 02, 2020, from https://towardsdatascience.com/the-3-best-optimization-methods-in-neural-networks-40879c887873

[63] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

Appendices

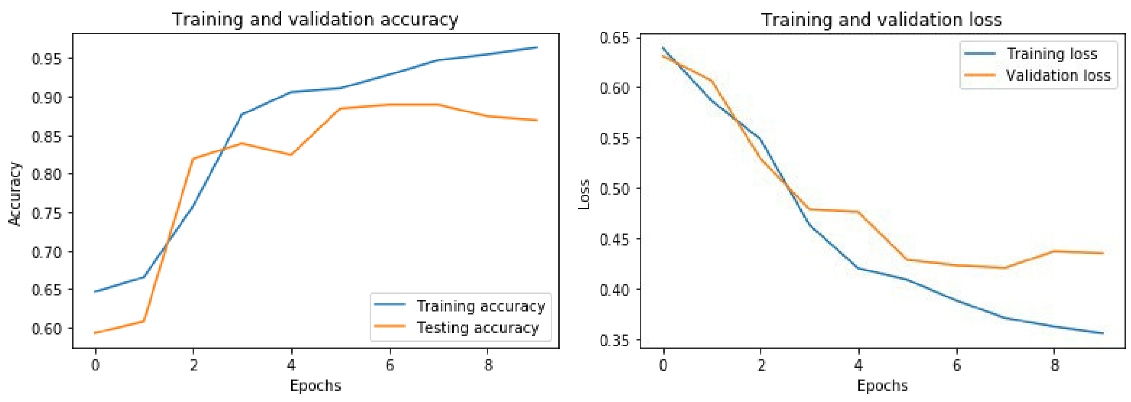


Figure A.1. Training and validation process for accuracy and loss at the CNN [SG,10,NS] test from Section 6.1.1.

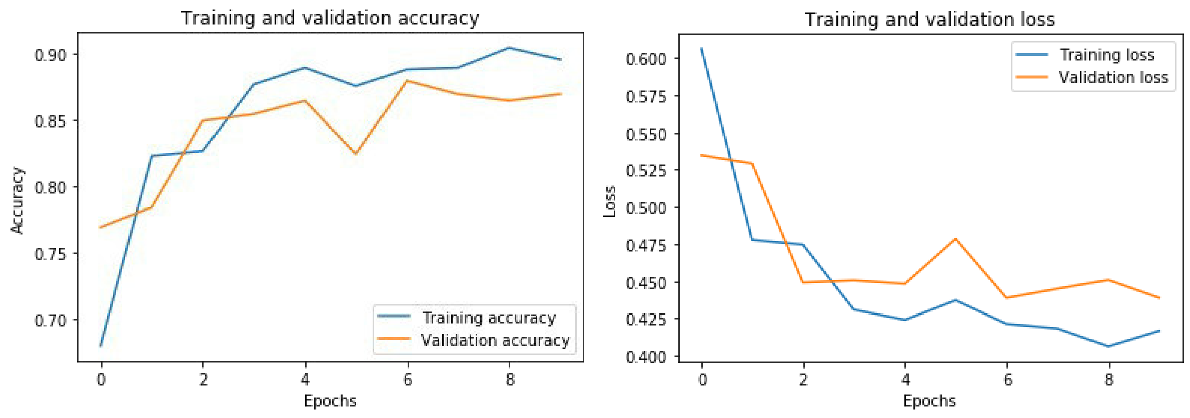


Figure A.2. Training and validation process for accuracy and loss at the LSTM [SG,10,NS] test from Section 6.1.1.

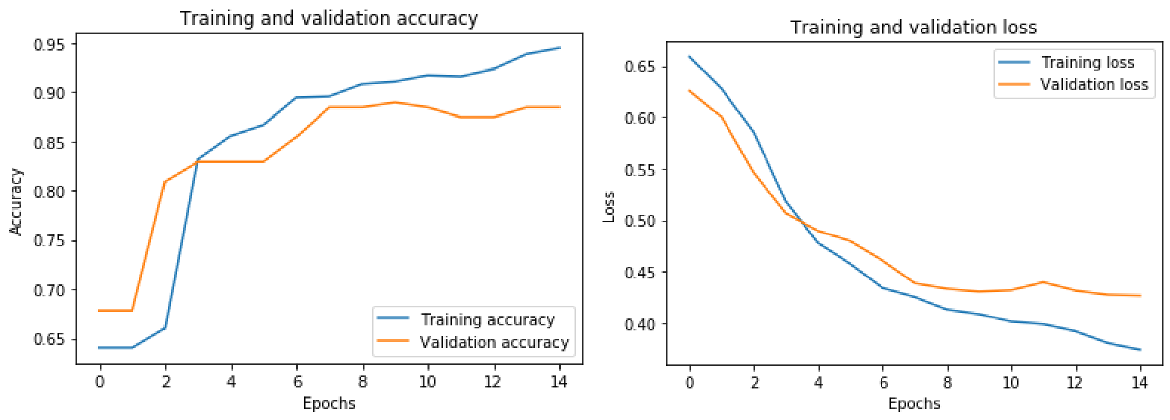


Figure A.3. Training and validation process for accuracy and loss at the BLSTM [SG,15,S] test from Section 6.1.1.

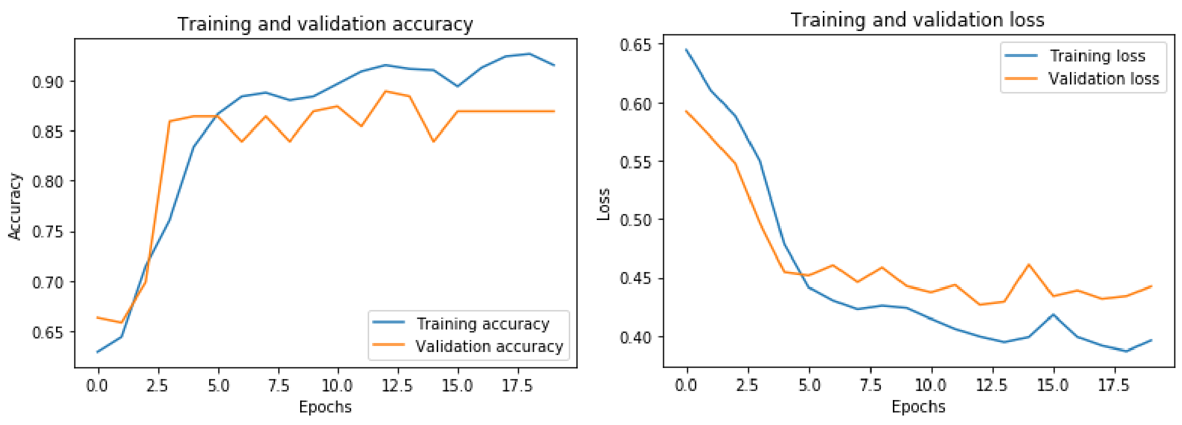


Figure A.4. Training and validation process for accuracy and loss at the GRU [SG,20,S] test from Section 6.1.1.

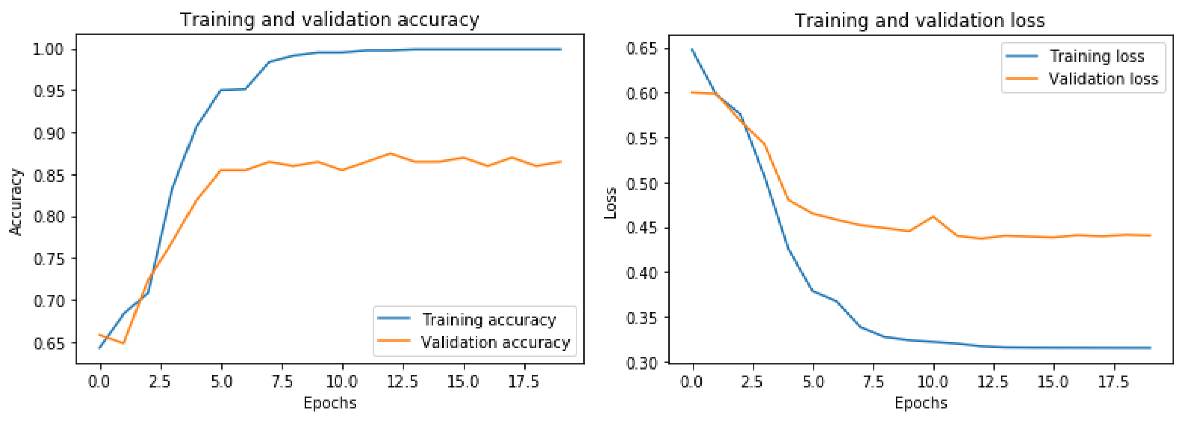


Figure A.5. Training and validation process for accuracy and loss at the CNN [SG,20,NS] test from Section 6.1.2.

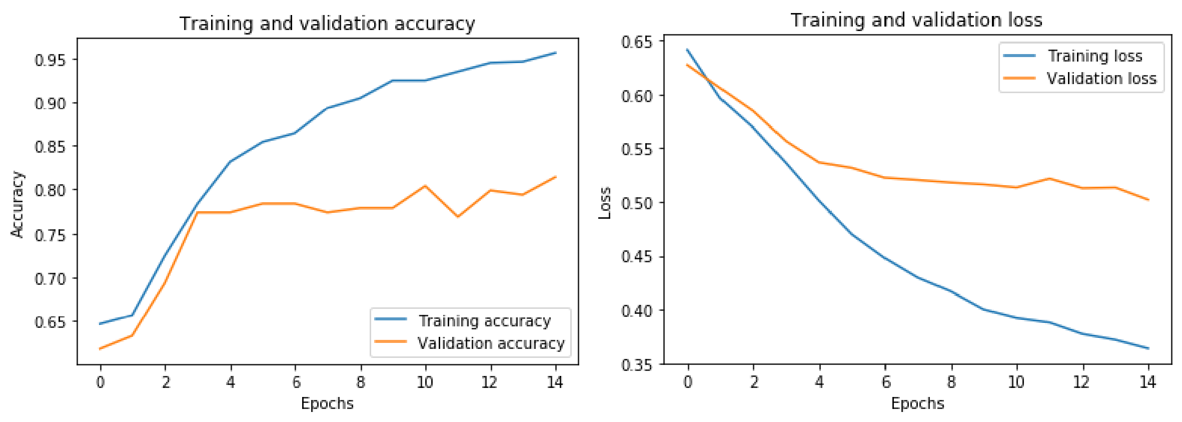


Figure A.6. Training and validation process for accuracy and loss at the GRU [15,S] test from Section 6.1.3.

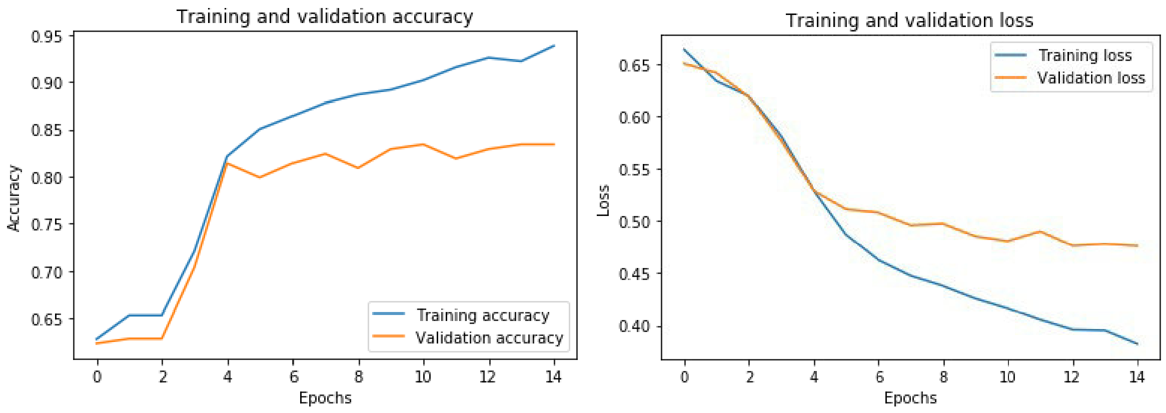


Figure A.7. Training and validation process for accuracy and loss at the BLSTM [15,NS] test from Section 6.1.4.