A Deep Learning Model for Detecting Sarcasm in Written Product Reviews

Nicole Schwarz

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2019

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, September 23, 2019

[illegible]

Nicole Schwarz

Contents

Preface

The basis for this research initially stemmed from my high interest in developing better methods of sarcasm classification. As the world moves further into the digital age, there will be a greater need to automate business processes, such as efficient product review evaluation according to their sentiment. It is my passion to contribute to this specific topic for future generations. In truth, I could not have achieved my current level of success without an active support group. First of all, my parents, who supported me with firm encouragement. And secondly, all the persons involved in the project's development, each of whom has provided patient advice and guidance throughout the research process. Thank you all for your unwavering support.

Abstract

The ability to reliably identify sarcasm and irony in a text can improve the performance of many Natural Language Processing systems, including summarization, sentiment analysis, etc. The existing sarcasm detection systems have focused on identifying sarcasm on a sentence level or for a specific phrase. However, often, it is impossible to identify a sentence containing sarcasm without knowing the context. This thesis aims at showing the possibilities of sarcasm detection in Amazon product reviews using a deep neural network. The data set for this project was acquired from the Github repository of Elena Filatova, who has already done research on collecting sarcastic Amazon reviews. As the task of irony detection yearns for corresponding algorithms, that can resolve contextual problems, the approach of using a deep learning model, containing a Convolutional Neural Network and a Long Short-Term Memory Network, was taken. Both can handle classifying texts highly depending on the context, as they have memory units to remember already learned words in a sentence. The results have shown that a deep neural network approach outperforms simpler models in their accuracy. Nevertheless, the best results have not yet been achieved due to the limit of data size.

Kurzfassung

Die Fähigkeit, zuverlässig Sarkasmus und Ironie in Texten zu erkennen, kann die Leistung vieler, maschineller Sprachverarbeitungssystemen verbessern. Dazu gehören die Zusammenfassung von Texten, Sentimentanalyse und weitere Methoden. Bereits existierende Sarkasmusdetektionen fokussieren sich auf die Erkennung von Sarkasmus auf Satzebene oder für eine spezielle Textphrase. Allerdings ist es oft unmöglich, einen einzelnen, sarkastischen Satz zu identifizieren, ohne den Kontext eines jenen zu kennen. Diese Arbeit beabsichtigt, die Möglichkeiten von Sarkasmuserkennung innerhalb Amazon Produktrezensionen mittels eines Deep Neural Networks offenzulegen. Die Daten für die Realisierung des Projekts wurden vom Github-Repository von Elena Filatova importiert, die sich in ihrer früheren Arbeit bereits mit dem Sammeln von sarkastischen Amazon-Produktrezensionen beschäftigt hat. Ironiedetektion verlangt nach passenden Algorithmen, die kontextuelle Probleme lösen können. Deshalb war der erste Denkansatz für die Implementierung des Projekts ein Deep Neural Network, das jeweils ein Convolutional Neural Network, sowie ein Long-Short-Term Memory Network umfasst. Beide Methoden können Texte möglichst effizient klassifizieren, die sehr vom Kontext des jeweiligen Textes abhängen, da beide über Speichereinheiten verfügen, um sich bereits gelernte Wörter aus dem Text merken zu können. Die Ergebnisse zeigten, dass Deep Neural Networks die Genauigkeit simplerer Modelle übertreffen können. Nichtsdestotrotz, bessere Resultate konnten durch den Mangel an Daten noch nicht erzielt werden.

Chapter 1

Introduction

Artificial Intelligence (AI) offers a broad range of areas to research in, *Machine Learning* (ML), *Deep Learning* (DL) and *Natural Language Processing* (NLP) being the main fields. Machine learning is defined as a set of statistical techniques for problem-solving, being applied to a wide variety of problems, such as vision-based research, fraud detection and price prediction, whereas Deep learning denotes a subarea of ML and defines an extension of Neural Networks. Deep learning algorithms are able to deal with NLP tasks as well but are mainly applied to vision-based classification problems. *Recurrent Neural Networks* (RNNs), *Convolutional Neural Networks* (CNNs) and *Long Short-Term Memory Cells* (LSTMs) belong in the field of deep learning. NLP is a confluence of artificial intelligence and linguistics. It involves the analysis of written language. Specific textual features could be investigated by performing Sentiment Analysis or information extraction on a text. The following figure illustrates the relationship between these three main categories. All three departments intersect and create a subsection of AI. As natural language processing has a strong linguistics component, the art of understanding language can be described as the ability to understand humour, sarcasm, the subconscious bias in text, and more. Once understood, a machine learning algorithm can be trained and therefore, automatically discover similar patterns statistically [53].

Investigating the complexity of AI and the power of current machine learning algo-

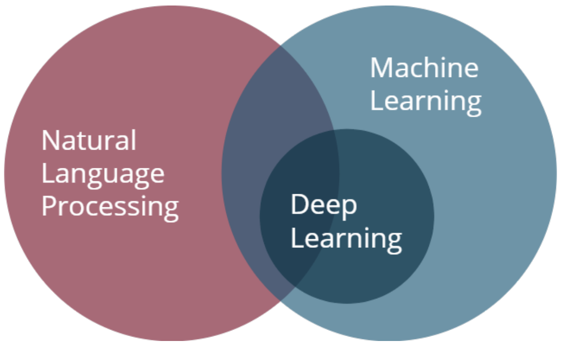


Figure 1.1: Illustration of the relationship between the Artificial Intelligence fields Machine Learning, Deep Learning and Natural Language Processing [53]

rithms, there seems to be one area of interest to be sparsely elaborated by now. This area is located within NLP and to be more exact, within Opinion Mining and Sentiment Analysis. Besides recognizing the either positive, neutral or negative polarity of a textual utterance, or identifying the emotion of an author of specific texts, there is rather high feasibility of detecting sarcasm in written paragraphs.

Sarcasm is defined by the online dictionary *Merriam-Webster* as "a sharp and often satirical or ironic utterance designed to cut or give pain" [51]. The term can be traced back to the Greek verb "sarkazein", which initially meant "to tear flesh like a dog". Especially on social media platforms, the possibility of being misunderstood by others is responsibly high. As the information on these platforms is primarily delivered in a textual form, audiovisual features of sarcastic utterances can not be perceived by the counterpart. Irony detection is applied within a specific area of Sentiment Analysis. Though, sarcasm itself is mainly relying on the tone of voice when being adopted within a majorly ordinary conversation and claiming the right set of features to recognize it within a text, seems to be challenging. Understanding sarcasm in online posts had been a significant factor holding back monitoring tools in the past years. Sarcasm detection in product reviews is primarily useful for automatic classification and sequencing of ratings as various goods could be more easily ranked according to seriously given critiques. This would enhance product representation for a numerous amount of businesses globally.

Some features of a speaker's tone of voice can be adapted within textual utterances as well. Colloquial, sarcastic writing style can involve an excessive usage of vowels in a word. Overexaggerating sentiment in a sentence could look the following way: "Oh, I just loooooove how you can speak about World War II in such a positive way!!!" This example also illustrates how applying exclamation marks can add to overstating the usual meaning of a written phrase. Further addressing the subject matter, people adhering to sarcasm usage can be unveiled by inspecting contradictory word or predicate handling in their sentences. "I admire your way of disrespecting my authority." gives the reader a line on ironic sentiment in the same way, as "admiring" something contradicts with "disrespecting" someone. Not to mention, that there are various persons not being able to understand sarcasm despite a sentence applying these features, depending on the age, the overall knowledge and the level of distinctive humour of the recipient.

Sarcasm detection investigates the pragmatic features of an online text on one side. Pragmatic features are, for instance, the usage of negations, as internal propriety of an utterance. Considering such an example can help to detect the disparity between the literal and the intended meaning of an utterance [20]. Another feature would be the appearance of positive or negative tempered emoticons. There, the adequacy of their usage can be discovered by analyzing the sentences' polarity. While there are various other pragmatic characteristics, the next areas of interest are the lexical or linguistic cues used in an utterance. Linguistic features can be related to word exposure and use, that represents the relative frequency of occurrence and use for specific words, such as spoken word frequency, semantic diversity, and age of acquisition [38]. Besides that, hashtags can also be used to determine the sarcastic meaning of a sentence. The existence of hashtags enables a simple collection of sarcastic data from websites such as *Twitter* or *Reddit*. On these platforms, the application of hashtags to express emotion is common and done regularly [31].

In machine learning, the procedure of sarcasm detection operates in the following manner. Once a machine understands a text, the insights of analyzing its components have to be unveiled. The simplest way to do this is by quickly looking at available data, and analyzing it the way a human would parse its features, such as using smiley faces for expressing emotions. Subsequently, a model can be built to perform such a task automatically. This work considers the boundaries of detecting irony, or sarcasm, within textual utterances, regarding specific features used, in online product reviews, or to be more exact, in reviews written on the online shopping platform and book trading company *Amazon*. Methods of achieving satisfiable results through machine detection are characterized and defined throughout this research.

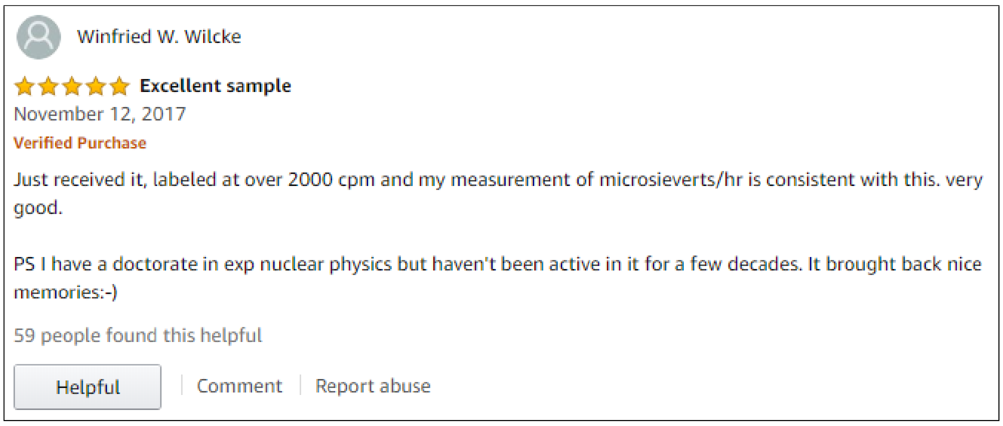
In the following few chapters, a detailed explanation of the process of detecting sarcasm in online product reviews is given. Chapter 2 discusses the related work in terms of sarcasm recognition, chapter 3 outlines the details of the research, including theoretical insight into the model structure, chapter 4 sets out the architecture, implementation and achieved results, and chapter 5 discusses the results and draws conclusions. The following two sections will focus on the definite purpose of the research, as well as on the chosen methodology.

1.1 Purpose of the Research

The ability to recognize irony is, for the most part, an odd feature of any human's intellect. Studies have shown, that the sagittal stratum, which is constructed out of fibre pathways (also called white matter tracts) recumbent on the sides of the brain, has the most significant impact on the human's ability to understand sarcasm [13][9]. For this reason, people who have suffered from strokes, and damaged this part of their control center, seemed to struggle in detecting irony or sarcasm in sentences [34]. The reason for trying so severe to remodel this part of the brain within a machine model is the following.

Sarcasm detection has gained reasonable attention in recent times. The change of categorizing media, according to sentiment, has consistently evolved through Sentiment Analysis. Despite, only common sentiments, such as happiness, sadness, anger or neutrality, are taken into account. One of a few categories missing is sarcasm or irony. The possibility to classify pictures or videos according to their sarcastic sentiment clears the track for businesses to offer a more detailed screening of their media's content.

Regarding the platform Amazon, an especially exciting usage of such an irony filtering system would directly affect the order of product reviews. Highly sarcastic written reviews could be culled, and more helpful comments would be shown as either highest or worst, honest product rating. The overall quality of recommended product reviews could raise. Taken two similar examples of ratings, shown in figures 1.2 and 1.3, of the exact item, a vast disagreement may be identified. Both ratings are taken from a product called "Uranium ore" for about $39.95. The two have an equal amount of stars given as the rating, which is five stars in this case. Figure 1.2 purely shows an honest rating of an experimental nuclear physicist, who has purchased the uranium and gave his sincere opinion about the behaviour of the product. The validity of the review is furthermore endorsed through the tag "Verified Purchase". Besides, figure 1.3 represents a well observable instance of a sarcastically-intended product review. It is indeed highly doubtful that a coworker would confuse uranium with liverwurst, as a consequence consume it


Winfried W. Wilcke

★★★★★ **Excellent sample**

November 12, 2017

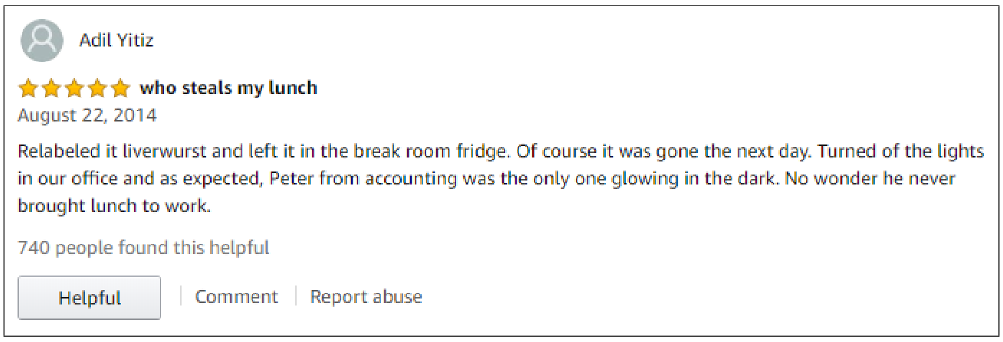
Verified Purchase

Just received it, labeled at over 2000 cpm and my measurement of microsieverts/hr is consistent with this. very good.

PS I have a doctorate in exp nuclear physics but haven't been active in it for a few decades. It brought back nice memories:-)

59 people found this helpful

Figure 1.2: Positive, earnest rating of a product on Amazon [63]


Adil Yitiz

★★★★★ **who steals my lunch**

August 22, 2014

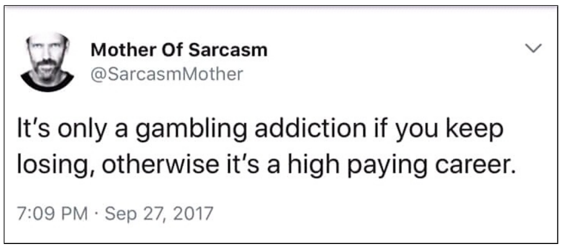
Relabeled it liverwurst and left it in the break room fridge. Of course it was gone the next day. Turned of the lights in our office and as expected, Peter from accounting was the only one glowing in the dark. No wonder he never brought lunch to work.

740 people found this helpful

Figure 1.3: Positive, sarcastically meant rating of a product on Amazon [64]

and therefore result in glowing in the dark. One will notice, that the verification tag for a valid purchase is missing on this review. Accordingly, apart from a distinction of the star rating and the meaning of a review, inspecting the verification of a purchase is another hint for the presence of irony. A realistic assumption about a product can most times only be given when there is actual proof of the product being bought from the critic.

Regarding an additional utilization of a sarcasm detection tool, the online platform Twitter is emphasized. Various posts and tweets, meaning online entries on a social media platform, are published on a daily basis. Several entries deal with the circumstance of being sarcastic. Unluckily, a few relate to topics such as bomb threats, assaults, and other sensitive topics. At that critical point, the secret service considers recognizing serious threats improbable. Automatic tools try to trace down the source of written dangers online, but none can accurately differentiate between serious or ironic texts [42]. This circumstance introduces a new gap of sarcasm detection, which is cybercrime control. Regarding such, the most straightforward way to get caught posting a bomb threat seems to be offered by the transparency of the social media platform Twitter. Figure 1.4 demonstrates one particular usage of sarcasm in an online text, where, for the sincere reader of it, it might not be entirely clear at the beginning, that this post is


Mother Of Sarcasm
@SarcasmMother

It's only a gambling addiction if you keep losing, otherwise it's a high paying career.

7:09 PM · Sep 27, 2017

Figure 1.4: Demonstration of a sarcastic post on Twitter [49]

meant to be sarcastic. There is no hint of sarcasm, as no overly emphasized words are used, nor is the hashtag `#sarcasm` provided. The obvious way to identify this tweet as sarcastic is by the overall nature of the author's profile. The profile name clearly states that all of the author's posts will be sarcastic. Nevertheless, there can always appear an exceptional case. In this case, it is rather hard to say, which features a machine would investigate to detect sarcasm. In the course of this work, a more detailed insight will be given into this issue.

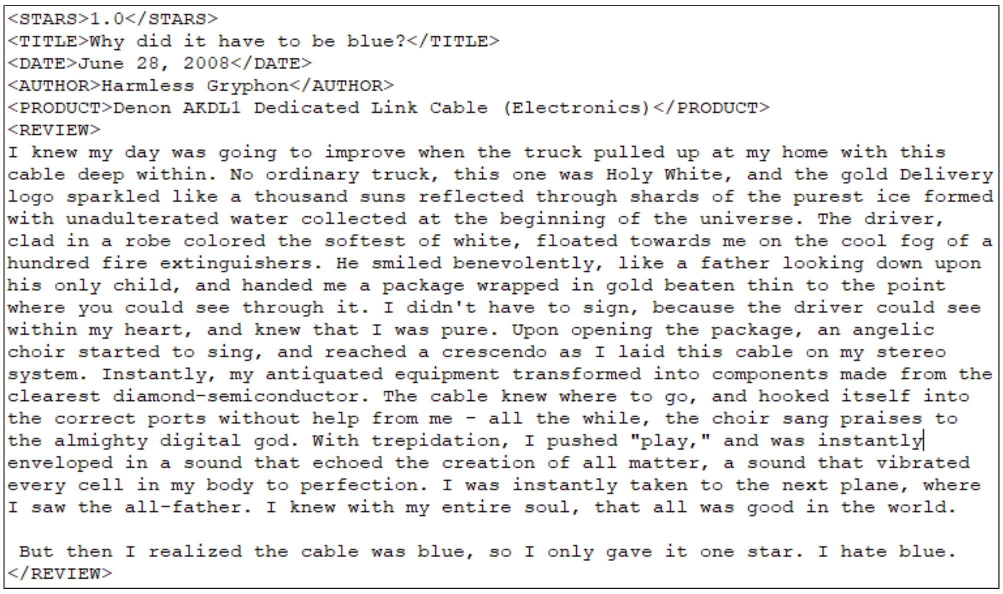
The major problem this work aims to solve, is in general, up to what extent it is possible to push the boundaries of sarcasm machine detection. Thus it appears that the overall definition of a boundary has to be tracked down, concerning sarcasm recognition in written text. This work fields the question if a deep learning model is capable of outperforming standard machine learning models, such as *Support Vector Machines* (SVMs) or *Logistic Regression* (LR) models, and if not, is there a need for a deep structural model at all. As of now, various approaches of sarcasm detection algorithms do exist, as further depicted in chapter 2. Those papers discuss the relevance of such a tool and define the accuracy of all their models in the end. As can be seen in chapter 2, the overall results of the models achieve around 79 to 82 percent accuracy for the irony detection task, majorly in tweets or reviews. The goal of this research is to result in higher accuracy than concurrent research outcome and exploit the limits of sarcasm recognition within a machine.

To fill the gap of this recently researched topic, distinct modest and deep learning structures shall be trained and evaluated to investigate performance fluctuation. For this reason, the methodology will be amplified in section 1.2.

1.2 Methodology

As already stated, a potential accuracy gain and the boundaries of sarcasm detection are the key areas this paper addresses. Therefore, thorough research compasses the evaluation of specific work, comprising sarcasm in text and particularly in product reviews. Jain, Ranjan and Baviskar [19] formerly achieved a satisfying percentage of accuracy in their research using light models, such as SVMs and LR structures. This work strives to outperform the simple approaches and gain higher accuracy by training a *Deep Neural Network* (DNN) architecture.

This work acts as a balanced combination of a qualitative and quantitative research method. Besides steeping into the broad spectrum of sarcasm detection within a text,



```
<STARS>1.0</STARS>
<TITLE>Why did it have to be blue?</TITLE>
<DATE>June 28, 2008</DATE>
<AUTHOR>Harmless Gryphon</AUTHOR>
<PRODUCT>Denon AKDL1 Dedicated Link Cable (Electronics)</PRODUCT>
<REVIEW>
I knew my day was going to improve when the truck pulled up at my home with this
cable deep within. No ordinary truck, this one was Holy White, and the gold Delivery
logo sparkled like a thousand suns reflected through shards of the purest ice formed
with unadulterated water collected at the beginning of the universe. The driver,
clad in a robe colored the softest of white, floated towards me on the cool fog of a
hundred fire extinguishers. He smiled benevolently, like a father looking down upon
his only child, and handed me a package wrapped in gold beaten thin to the point
where you could see through it. I didn't have to sign, because the driver could see
within my heart, and knew that I was pure. Upon opening the package, an angelic
choir started to sing, and reached a crescendo as I laid this cable on my stereo
system. Instantly, my antiquated equipment transformed into components made from the
clearest diamond-semiconductor. The cable knew where to go, and hooked itself into
the correct ports without help from me - all the while, the choir sang praises to
the almighty digital god. With trepidation, I pushed "play," and was instantly
enveloped in a sound that echoed the creation of all matter, a sound that vibrated
every cell in my body to perfection. I was instantly taken to the next plane, where
I saw the all-father. I knew with my entire soul, that all was good in the world.

 But then I realized the cable was blue, so I only gave it one star. I hate blue.
</REVIEW>
```

Figure 1.5: Extract of Elena Filatova's Sarcasm Corpus showing a sarcastic entry [12]

a more qualitative approach of informing about the specific task of recognizing such within product reviews is followed. A vital difference to sarcasm recognition, in general, is given by the presence of star rating, particularly in product ratings or the tag of the purchase being verified. During this research, the star rating is carefully looked at, as the utilized data, further described in section 1.2.1, does not contain any information about the purchase validity. The guarantee of the star rating providing a satisfying solution to sarcasm detection in a review is constituted by the case, that a low number of stars given, alongside an overall positively written review, appears to indicate the presence of irony. Potential throw-outs that do state a positive sentiment in the review have a five-star rating as well but do indicate sarcasm, are labelled in the data set as sarcastic anyway and do therefore not affect the outcome on a large scale.

1.2.1 Sarcasm Corpus

The data retrieved for the specific task of sarcasm detection is obtained by the research of Elena Filatova [12] during her studies at the Fordham University in New York. The corpus comprises a multitude of Amazon products, separated into sarcastic (or ironic) and regular reviews. The labelling of the reviews was accomplished by workers of the service *Amazon Mechanical Turk* (MTurk). This data set is chosen because of the extensive labelling and the presence of specific information, such as the star rating, which may give additional certainty for sarcasm detection. In figure 1.5, an example of a sarcastic data entry can be seen. The corpus was generated in 2012, and although the work is somewhat obsolete, the data collected is still useful for training and testing a model for the task of detecting sarcasm. The quirks of sarcasm may not appear to change over time. Thus the validity of the deep neural network will not be affected.

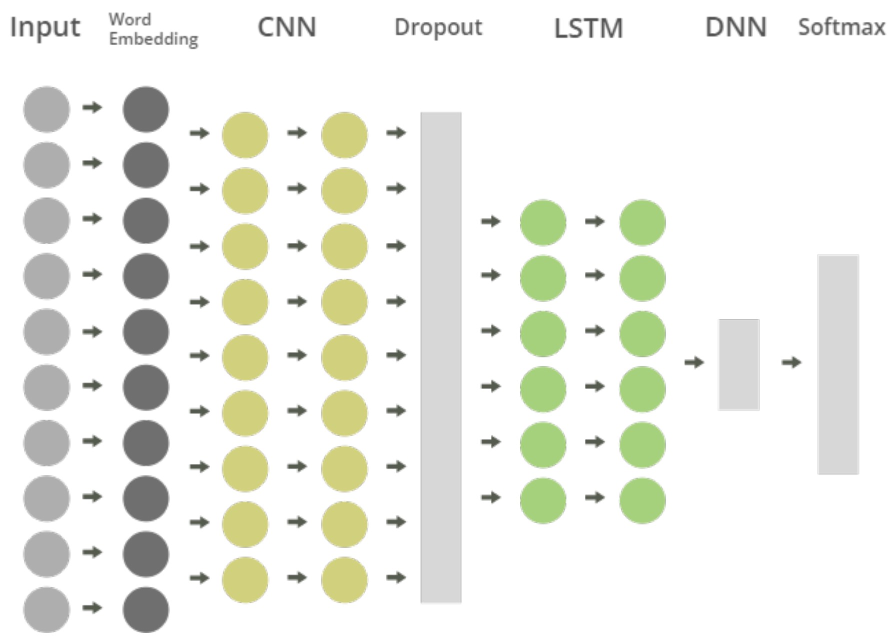


Figure 1.6: Elementary structure of the sarcasm detection DNN [14]

1.2.2 Deep Neural Network

After obtaining the corpus's reviews, a next step is to categorize them as sarcastic or non-sarcastic. Sarcasm can further be indicated by changing sentiment polarity between sentences. Thus the model for categorizing the reviews shall be time-sensitive. Ghosh and Veale [14] have conducted a research on building a deep neural network for sarcasm detection on Twitter. Figure 1.6 shows a rough structure of their model. As can be seen in the illustration above, the first layer of the model is an input layer, strictly followed by one of the main components in the structure, the word embedding layer. Word embeddings are a specific type of document vocabulary representations. They are capable of capturing the context of a word in a document. The same applies to semantic and syntactic similarity and the relation with other words. Loosely speaking, they represent a particular word in a vector [48]. Next, there is a two-layer CNN, being responsible for reducing the frequency variation through convolutional filters and extracting discriminating word sequences as a composite feature map for the trailing LSTM units. In between, there is a dropout layer, which shall avoid the overfitting problem to an extent. The output of the LSTM layer is further passed to a fully connected dense layer. It produces a higher-order feature set based on the LSTM outcome, which is readily separable for the desired number of classes. Finally, a softmax layer appears at the bottom of the network. Using this deep configuration and comparing it to lightweight ones, favorable results are expected.

1.2.3 Results

The results achieved during this work, using a deep learning model composed of several neural network layers, will be compared to actual results of more trivial machine learning models, that state about 79 to 82 percent accuracy. If the formerly trained deep neural network is then later evaluated, a difference in the percentage rate shall be identified. Besides, the aggregate disparity of the different models shall be at least one percent, to imply a drastic change in performance, therefore either innovation or stagnation. The resulting limits of sarcasm detection, by the machine as well as by the human, will then be compared to other research papers, dealing with the boundaries in both parties. Rather than conducting an extensive survey process totalling this research study, the existing research outcome will be used to gather information about reasonable limitations of sarcasm detection by humans. Notwithstanding that a DNN model may be an exorbitant choice for this problematic task, issuing typical machine learning complications such as overfitting or possessing sparse data, and simpler models may yield improved results, alternative findings might be revealed at the end of this research.

Chapter 2

Related Work

The task sarcasm detection has been tackled a considerable amount of times in the ML sector. Traditional models employ discrete features to address the task. This paper investigates the possibilities of implementing deep neural network sarcasm detection systems. In particular, the effect of Convolutional Neural Network and Long Short-Term Memory algorithms is examined.

Research shows that sarcasm detection has become a requirement for various businesses. *The Telegraph* [29] states that online abuse has become a scourge of social media companies. As of now, Artificial Intelligence aims to understand the context of posts in order to avoid extremist messages, bullying or national threat. On social media, every textual post tends to communicate a particular feeling, be it expressing an emotion itself or simply handing out one's opinion. Especially in the last case, various forms of sarcasm are used to converse by messages. Deep Learning was first utilized to understand distinct sentiments from a text. Since then, Natural Language Processing systems apply sentiment analysis for classifying text according to positive and negative polarity [52]. Sarcasm mostly delivers vague signals to language processing systems and contextual meaning enacts a major part in detecting such.

The following sections will discuss sarcasm detection in various areas, beginning with the presence of sarcasm on social media, to the actual problem of its occurrence in product ratings. Different approaches will be tackled, and the problems of sarcasm will be elaborated.

2.1 Sarcasm Detection in Social Media

Sentiment detection plays a ground role in natural language processing. Though joy, sadness, fury and enthusiasm can be adequately categorized within texts, sarcasm detection remains a challenging aspect in this field. A paper written by MIT researchers [11] informs about their created algorithm, that can translate the tone of social media posts. Therefore, it can understand when someone is sarcastic. This detection is especially useful because of its ability to consider emojis. The emotional subtext is understood, and sarcasm usage can be identified. Their application is called *DeepMoji.* In their paper, they analyzed 1.2 billion tweets to understand how 64 favored emojis are used to convey meaning. The system was first trained to recognize sadness, happiness or fun. Given the context, the system was applied to understand and interpret the feeling behind texts. The accuracy of 82 percent correctly perceived sentiment demonstrates the reliability of this algorithm, notably because it was outperforming human volunteers having 76 percent accuracy state. The paper describes a variant of the LSTM model. DeepMoji handles an embedding layer of 256 dimensions, projecting each word into a vector space. A hyperbolic tangent activation function enforces a constraint of individual embedding dimensions being within $[-1, 1]$. Two bidirectional LSTM layers with 1024 hidden units and a superincumbent attention layer using skip-connections complete the deep neural network. The attention mechanism lets the model decide on the importance of each word by weighing them when constructing the depiction of the text.

Bamman and Smith [2] conducted a research in the field of contextual sarcasm detection on the social media platform Twitter. Though sarcasm detection as an everyday text classification task provides considerable accuracy, it should not entirely ignore the context beyond a sarcastic utterance's surrounding. They decided to device the available features into four classes comprising tweet features, author features, audience features and environment features. Tweet features included linguistic indicators, such as word n-grams, and sentiment-related aspects. The author features mainly comprised historical data of the author and profile information. Audience features described the ones related to the interaction of the author with the audience. Lastly, environmental features characterized the interaction between a tweet, and the particular tweet it is responding to. The experimenters discovered that the tweet's author historical striking terms were the most significant indicator of sarcasm, with an accuracy of 81.2 percent, using a LR model. In contrast to that, single tweet features score with 75.4 percent class accuracy. Nevertheless, a combination of both feature classes results in an accuracy percentage of 84.9. Combining all of the features did not significantly increase accuracy.

Liebrecht et al. [26] worked on an entirely different approach during their work at the Centre for Language Studies of the University in Nijmegen, Netherlands. The researchers collected a training corpus of 78 thousand tweets in the Dutch language containing the hashtag `#sarcasme`, the Dutch word for sarcasm. The data collection for this task resulted in a set of 77,948 tweets. Entries were tokenized and stripped of punctuation. A ML classifier was trained on the harvested examples and applied to a test set of a day's stream of 3.3 million Dutch tweets, of which 135 carried the hashtag `#sarcasme`. Using this trained system, they were able to detect 101 (75%) unmarked tweets out of 135 explicitly marked ones. In their paper they claim that sarcasm is signalized by hyperbole, using intensifiers and exclamations. They hypothesize that explicit markers, hashtags in this particular case, are the digital extralinguistic equivalent of people's nonverbal expressions when conveying sarcasm. In their work, they made use of word uni-, bi- and trigrams as features for training the system. As the major classification algorithm, they utilized the *Balanced Winnow* method [27]. This particular algorithm seems to offer state-of-the-art results in text classification and produces weights, that can be used to, for instance, inspect the highest-ranking features for a class label. The research paper also argues that the detection of sarcasm is crucial for the development and refinement of sentiment analysis systems, but at the same time a conceptual and technical challenge. To evaluate the outcome, the group performed two evaluations. The first one focusing on 135 tweets being explicitly labelled, posted on February 1, 2013, was measured by the *True Positive Rate* (TPR), the *False Positive Rate* (FPR), also known as recall, and the *Area Under the Curve* (AUC). The AUC is an evaluation metric, argued to be more resistant to skew than the general F-score, due to using TPR instead of precision [10]. The researchers found to reach an AUC value of about 0.79 when using their trained ML system on their dedicated Twitter corpus.

Maynard and Greenwood [28] follow an almost identical approach, besides identifying multiple hash marks. In addition to the hashtag `#sarcasm`, indicators such as `#notreally` are determined and added to their list of sarcasm indicators. Due to these constraints, they developed rules for recognizing sarcasm using various hashtags. For this reason, the hashtag `#sarcasm` does not have to be present in every tweet, a combination of other hash marks, such as `#great #notreally`, can be identified as having a sarcastic tone as well. Using their method, they were able to achieve a precision of 74.58 percent.

Lastly, Bharti et al. [3] claim that opinions posted on social media can be classified as positive, negative and neutral, and that a positive sentiment can further be classified as actual positive or sarcastic. The same scheme applies to negative sentiment. For training, 50,000 tweets were collected containing the sarcasm hashtag from Twitter. To test their first algorithm, which is a *Parsing Based Lexical Generation Algorithm* (PBLGA), 1,500 random tweets containing the hashtag, and 1,500 random tweets without it, were used. To test the second algorithm, called the *Interjection Word Start* (IWS) algorithm, 1,000 tweets containing the hashtag, and 2,500 tweets without the explicit marker, were used. In this case, both sets start with an interjection word. After preprocessing the data and training the algorithm, the researchers achieved a precision of 0.89 percent in case of the PBLGA containing sarcastically labelled tweets. Regarding the IWS algorithm, they achieved a precision percentage of about 0.85, containing tweets with the sarcasm hashtag.

2.2 Sarcasm Detection in Product Reviews

Researchers at The Hebrew University in Jerusalem, Israel [39] worked on a semi-supervised recognition system for sarcasm in online product reviews. Tsur and Rappoport claim that the identification of sarcastic reviews can "improve the personalization of content ranking and recommendation systems". Their model *SASI* is short for "Semi-supervised Algorithm for Sarcasm Identification". The algorithm employs a novel, semi-supervised pattern acquisition model for identifying sarcastic patterns on one side, and a classification algorithm classifying each sentence on the other. The data used was an extensive collection of Amazon user reviews, covering 66,000 of such, for different types of products. A set of features further used in feature vectors was extracted from the given, labeled sentences. They utilized the syntactic and pattern-based features. In addition to the pattern-based features, punctuation-based features were considered as meaningful. The researchers approached a k-nearest neighbors (kNN)-like strategy, to assign a score to new examples in the expanding test set. The paper outlines the exploitation of meta-data provided by Amazon, namely the star rating each reviewer is obliged to provide. In respect of negative product ratings, their baseline classifies as sarcastic those sentences that exhibit strong positive sentiment. Evaluating pattern acquisition efficiency, they achieved 81 percent in 5-fold cross-validation.

Buschmeier et al. [5] analyzed the impact of features for classifying irony in product reviews. The German researchers modelled the task of irony detection as a supervised classification problem. They focus on the impact analysis of different features by investigating what effect their elimination has on the performance of the approach. To estimate the level of irony in a sentence, they investigate the imbalance between the overall polarity of words in the review and the star-rating given for the particular product, as a first feature. The hyperbole [25] indicates the occurrence of a sequence of three positive or negative words in a row. The next feature describes, that up to two consecutive adjectives or nouns in quotation marks have a positive or negative polarity. Likewise, if there exists a span of up to four words containing at least one positive (negative) but no negative (positive) word and ends with at least two exclamation marks or a sequence of a question mark and an exclamation mark, another feature can be identified [7]. They investigate the same feature, but with a sentence being surrounded by an ellipsis, instead of ending with punctuation. Another feature comprises the presence of an ellipsis, followed by multiple exclamations or quotation marks. Analogously, a merge of both is examined, which is called the punctuation feature. Similarly, interjection features such as “wow” or “huh”, laughter and emoticons are considered as features. In addition, they use each occurring word as a feature (“bag-of-words”). For training a ML algorithm, they used the corpus of Elena Filatova [12], further discussed in section 2.3. For the machine learning algorithm itself, the group considered Naïve Bayes, SVMs (with a linear kernel), LR, decision tree and random forest classifiers. In the end, they were able to achieve 74.4 percent accuracy on the identification task using LR with all features, including the star-rating.

Jain et al. [19] set their focus on sarcasm detection in product reviews as well. Again, the corpus of Elena Filatova [12] was used to train and test the machine learning algorithm. Their training data set was composed of three hundred sarcastic and three hundred regular entries. For testing, they used a hundred entries of each category. For preprocessing the data, the rating of the product is scaled, and the review text is tokenized accordingly. Regarding feature engineering, the research group considered sentiment, punctuation, each part or sentence of the review, word unigram and bigram features, and contextual features (such as the absolute difference of the rating of the product and its normalized sentiment score). For the classification task, they used Naïve Bayes, a *Multi-layer Perceptron* (MLP) and an SVM. They distributed their data set into training and testing set, by splitting it into six hundred reviews for training and two hundred reviews for testing. Each category, whether sarcastic or regular, is present in equal proportion within both sets. After extracting the features of each review, the data are trained on the various classifiers mentioned before, and the subsequent results are observed. By utilizing the SVM classifier, an accuracy of 81.5 percent could be achieved.

2.3 Sarcasm Detection Data Source

Elena Filatova [12] seems to be the most suitable counterpart when searching for a sarcasm corpus comprising product reviews in detail. She worked on creating a sarcasm corpus, in which its entries serve to be utilized for in-depth study of different linguistic phenomena that make a text utterance sarcastic or ironic. In contrast to already existing corpora of Amazon product reviews, her data comprises text documents rather than separate sentences, as she believes that providing a context is of vast use for learning patterns of text utterances containing sarcasm. For better understanding the phenomenon of sarcasm, pairs of Amazon product reviews were collected, where both reviews were written for the same product, but express distinct sentiment.

Filatova claims, that in many cases a stand-alone text utterance, such as a single sentence can not reliably be judged as sarcastic or not without the surrounding context. For this reason, the question “Where am I?” can not be categorized as sarcastic unless it is known that it is written within a product review of a GPS device. She argues that in the absence of a strict definition of irony, it is impossible to train experts who could reliably identify irony. However at the same time, she is confident that the task of irony detection seems to be quite intuitive.

For creating a sarcasm corpus, the Amazon service MTurk was assigned the task to collect data, label it accordingly and therefore ensure the quality of it. The data set consisted of whole documents and documents pairs of product reviews published on Amazon. Therefore, the corpus can be used for identifying sarcasm on document and text utterance level. An entry itself contained the link to the product, the number of stars assigned, and further components. A dedicated list of instructions has been provided to the MTurk workers, describing the review structure, contents and shape. Additionally, for every entry a permalink was obtained, to retrieve the text of the product review together with supplementary information, such as the number of stars assigned to the review. Likewise, the labeling of the reviews was needed to test Filatova’s hypothesis on whether people can reliably distinguish between irony and sarcasm. Labeling was accurately done for 1,000 pairs of Amazon product ratings, excluding a few reviews not meeting the requirements. The size of the texts varied from a phrase to a whole document. After the data cleaning procedure, the corpus remains with 1,905 documents, most of the reviews being ironic - non-ironic pairs.

In the second step, data quality control was performed by letting different workers guess the number of stars assigned to a product by the review author. The goal was to identify a clear image of which product ratings have been seen as clearly sarcastic or regular. Lastly, the reliability of every annotator was computed and rated based on the quality of their previous work. After this particular action, the corpus ended up having 437 Amazon reviews containing irony, and 817 ones being regular/non-sarcastic. As expected by the researcher, the majority of the sarcastic reviews are written by people assigning low scores to the reviewed products, represented by 59.94 percent of the audience who submitted a one-star review. Regular reviews were rated with five stars (about 74 percent). Thus, Filatova assumes that it is easier to find sarcastic reviews among those who have assigned low scores to products, and on the other hand, that the chance to find an ironic review among the reviews that assign high scores to a product is low.

		Number of reviews with				
		1*	2*	3*	4*	5*
sarcastic	437	262	27	20	14	114
regular	817	64	17	35	96	605

Table 2.1: Number of data entries and their distribution amongst the star rating according to their label [12]

Appreciating the work of Filatova, the collected entries in the corpus fit well for training a dedicated deep learning algorithm. She confirmed that the hypothesis of sarcasm is often used in those reviews that give a negative score to the product, as well as, that the presence of irony in a product rating does not affect the readers' understanding of the product quality.

2.4 Word2vec

Mikolov et al. [30] proposed two novel model architectures for computing continuous vector representations of words from vast data sets. They also mention that the quality of these representations is measured in a word similarity task. Using their model, they observed substantial improvements in accuracy in much lower computational cost. They compare different model architectures, where they first defined the computational complexity of a model as the number of parameters that need to be accessed to train the model fully. The first architecture proposed is the *Continuous Bag-Of-Words* (CBOW) model, which they call a *Bag-Of-Words* (BOW) model, as the order of words in history does not influence the projection according to their research. The second architecture is similar to CBOW, but in contrast to predicting the current word based on a context, it tries to maximize classification of a word based on another word in the same sentence.

Training complexity of the CBOW model is then

$$Q = N \times D + D \times \log_2(V). \tag{2.1}$$

At the input layer, N previous words are encoded using 1-of-V coding, where V is size

of the vocabulary. The input layer is then projected to a projection layer P that has dimensionality $N \times D$, using a shared projection matrix. With binary tree representations of the vocabulary, the number of output units that need to be evaluated can go down to around $\log_2(V)$. This model, as unlike a standard bag-of-words model, uses continuous distributed representation of the context.

The training complexity of the second, Skip-gram architecture is proportional to

$$Q = C \times (D + D \times \log_2(V)), \tag{2.2}$$

where C is the maximum distance of the words.

Results have shown that when they train high dimensional word vectors on a large amount of data, the achieved result vectors can be used to answer very subtle semantic

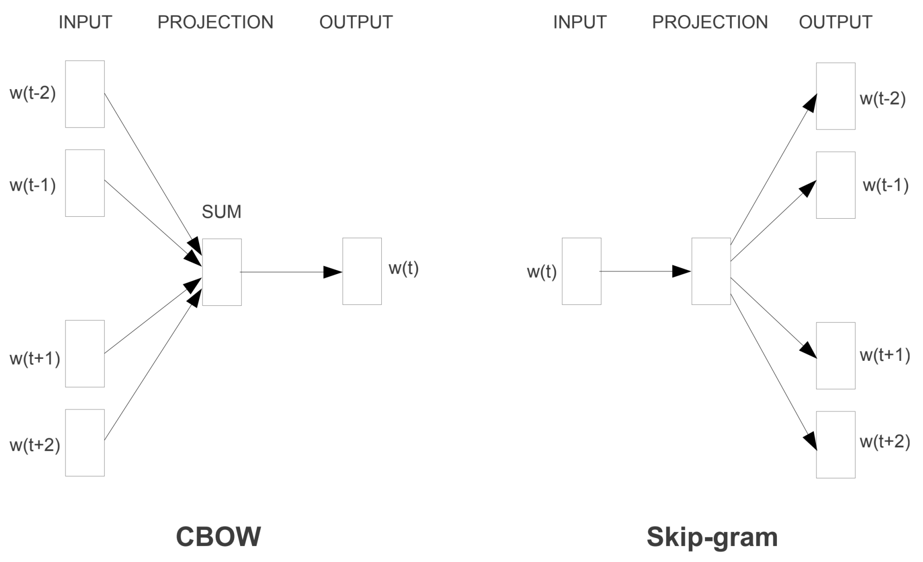


Figure 2.1: The novel model architectures by Mikolov et al. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word [30].

relationships between words. This involves examining the relationship of a city and the country it belongs to. An example of such a relation would be, for example, to classify “France and Paris” as “Germany and Berlin”. The architecture of both models can be seen in figure 2.1. The researchers defined a comprehensive test set containing five types of semantic questions, and nine types of syntactic questions, to measure and verify the quality of the word vectors. Overall, they used 8869 semantic and 10675 syntactic questions. The questions in each category were created in two distinct steps. At first, a list of similar word pairs was created manually, followed by a large list of questions formed by connecting two word pairs. For example, they made a list of 68 large American cities and the states they belong to and formed about 2.5K questions by picking two word pairs at random. The data were composed only of single token words, multi-word entities, such as “New York”, were not present.

They evaluated the overall accuracy for all question types as a whole, and for each question type separately. Every question was assumed to be correctly answered if the closest word to the vector computed, using the above method, is exactly the same as the correct word in the question. As the models have not yet possessed of input information about word morphology, reaching 100 percent accuracy was likely to be impossible. The researchers claim that further progress could be achieved by incorporating information about the structure of words, especially for the syntactic questions.

These word vector representation models are part of the implementation of *Word2vec*. Word vectors with such semantic relationships are already used to improve many existing NLP applications, dealing with machine translation, the retrieval of information or the answering of questions in a system. Their work shows that word vectors can be successfully applied to an automatic extension of facts in Knowledge Bases, and also for verification of correctness of existing facts.

2.5 Problems in Sarcasm Detection

Ghosh et al. [15] consider the fact that there is a low success rate in detecting sarcasm when investigating it in an isolated sentence. In their work, they observe the role of conversation context for detecting sarcasm in online interactions. In general, context is indispensable for the efficient recognition of sarcasm in a text. The researchers investigated, that LSTMs that can model both, the context and the sarcastic reply, achieve better performance than LSTM networks, that read only the reply. In particular, they reviewed conditional LSTM networks, as well as LSTM networks working on sentence-level attention, which achieved significant improvement of around 6 to 11 percent in f1-score. To address another problem, describing the finding of significant text parts in regard to the context. For this reason, a qualitative analysis of attention weights produced by the LSTM models with attention was presented and discussed with human annotators. Attention-based models are able to identify the inherent characteristics of sarcasm, such as context incongruity.

Wicana et al. [41] claim that currently, researchers largely misled between the linguistic properties of sarcasm and the linguistic properties of sarcasm. Kreuz and Glucksberg [23] state a theory, in which sarcasm and irony seem to be similar, yet different in that sarcasm has a specific victim who is the target of a particular mocker. In irony, the generality is approached, where no one, in particular, is the victim. Amir et al. [1] cling to the opinion that sarcasm requires a common ground between the distinct parties, represented by the author and the target, to be understood. Considering this qualifier, the researchers tried to compute the probability of a relationship between users and the content it produces by using a mathematical function. Wicana et al. argue that the inability to make one useful framework of sarcasm detection resides in the fact that sarcasm itself is extensible to be used for any situation, platform and condition. They speculate about taking a full step towards this circumstance and re-assessing the definition of sarcasm itself because the recent definition of sarcasm and irony is different in the first place.

Regardless of already existing work, the act of explicitly recognizing sarcasm in product reviews is not approached using a deep neural network algorithm. Considerable research has already been done regarding the capability of realistic sarcasm detection algorithms. Nonetheless, there is just one scientific article addressing the specific problem of detecting sarcasm in reviews combining sentiment and rating of such. Considering this aspect, an unprecedented approach of applying a deep learning model for sentiment analysis in Amazon reviews will be adopted. The first approach comprises a CNN, as well as an LSTM network. As far as it has been researched, context is a crucial aspect of sarcasm detection in text. Word embeddings (or word vectors) are provided as input. Those vectors generally represent the sentences themselves. Furthermore, max pooling is applied to the feature maps to generate features. A fully connected layer, followed by a softmax layer outputting the final prediction, completes the deep neural network.

Chapter 3

Details of the Research

To resolve the research questions enclosed by this work, a principle understanding of the areas it addresses is needed. This work is located in the largely attractive area of AI, where applications are implemented for letting a machine think for itself. The current chapter aims to give an understanding of the basic principles applied in the implementation part of the research.

3.1 Artificial Intelligence and Machine Learning

Both of the areas, machine learning and deep learning, reside in the supercategory AI. The online dictionary Merriam-Webster defines artificial intelligence firstly as “a branch of computer science dealing with the simulation of intelligent behavior in computers” and secondly as “the capability of a machine to imitate intelligent human behavior” [50]. Those two very similar definitions do not vary much in their meaning. Both describe the possibility of intelligent human behaviour being imitated by a machine.

3.1.1 Artificial Intelligence

Artificial intelligence enables machines to learn from experience, to process recently obtained information and to handle tasks themselves, that require human-like rationality. Most applications of AI nowadays seem to be based on deep learning and NLP, where an extensive quantity of data is processed and specific patterns can be identified by machines. Artificial intelligence was first characterized in 1956, but has these days just acquired greater importance, due to evolving algorithms and the improvement of computational power. The revolution of AI over its history begins in the years 1950 to 1970, when neural networks drummed up enthusiasm for“thinking” machines. From the 1980s to 2010, machine learning gained importance. Today, the breakthroughs in deep learning tend to boost the interest in AI algorithms even more. AI automates the learning through repetition and the discovery of data. It seldom acts as an isolated application, instead, it upgrades existing implementations to be more intelligent. Multiple hidden neurons allows the analysis of a greater amount of data and of greater depth [44].

Forms of Artificial Intelligence

AI is on one hand classified into two distinct categories, namely the "weak AI" and the "narrow AI". The weak AI addresses systems, which are implemented and trained for a specific task, such as virtual personal assistants. The other form, also known as the common AI, describes a system possessing universal human cognitive skills, for it to be able to find a solution to any occurring problem it may not know about. The *Turing test*, contrived by the mathematician Alan Turing in the 1950s, serves as a method to determine the ability of a computer to think like a human [58].

On the other hand, there is the classification approach of the assistant professor for integrative biology and informatics Arend Hintze. He categorizes AI into four diverging types. The first area describes reactive machines, such as the chess-playing mechanism by IBM. They do not possess their own mind and are not able to evoke experience from the past. The second type has limited storage and can use it, meaning past experiences, to finalize decisions for the future. Some of the decision making in automotive driving cars works in this manner. The second last type describes the native theory, a psychological term referring to the understanding, that others might have their own persuasions, desires and purposes, influencing decisions. Hintze refers to this type of AI as a future type, as it might not exist so far. The last category of AI is self-awareness, where systems hold a certain level of consciousness to understand their current state and use this information to witness how others feel [58].

Applications of Artificial Intelligence

Applications of artificial intelligence vary from implementations in the general AI sector, to more enhanced ones in the machine learning branch. Automation is the process where a system acts and learns automatically. The *Robotic Process Automation* (RPA) can be programmed in such a manner, that it can undertake high volume tasks, which would normally be conducted by humans. It can automatically adjust to changing situations. ML is the science of making a computer do a task, without explicitly programming such for it. Deep learning is a subcategory of machine learning and might be seen as the automation of predictive analytics. Forms of machine learning applications can be supervised learning, unsupervised learning or reinforcement learning, which are further described in section 3.1.2 [58].

Machine Vision (MV) is the science of giving sight to a machine. It principally works on visual data or information, collected by cameras, analog-digital conversion and signal processing. It is deployed in several applications, beginning with signature identification to medical image processing. Lastly, there are *Pattern Detection* (PD) and the area of robotics. PD is a branch of machine learning concentrating on the identification of patterns in data. Robotics is applied in the area of engineering, bringing the development and production of robots into focus. Such robots are utilized in the automotive sector or astronautics, to manage work which might be too difficult for a human to undertake. Most notably, robots for social interaction appear to be strongly worked on [58].

3.1.2 Machine Learning

Machine learning is defined as a method of data analysis that automates analytical model building. It is further defined as a branch of artificial intelligence based on the idea that systems can educate from data, identify patterns and make decisions with minimal human intervention. The main difference of machine learning to other learning systems is that the principal goal is to understand the data structure and fit theoretical distributions to the data that are well understood. Due to the utilization of an iterative approach to learn from data, the learning of machine learning systems can be easily automated. The data are long enough fed to a model until a robust pattern is found. Besides other methods, two of the most widely adopted machine learning methods are supervised learning and unsupervised learning, further described in the sections below [45].

Supervised Learning

Regarding supervised learning, data sets are trained using labeled examples. This means, there exists input, where the desired output is known. In the case of this work, supervised learning is approached due to the existing data set by Elena Filatova [12]. There, a piece of data is labeled as ironic, whereas the other piece of data is labeled regular. The learning algorithm receives a set of inputs along with the corpus and learns by comparing its actual output with all outputs to find sarcastic reviews. Through methods like classification, regression, prediction and gradient boosting, the learning method uses a pattern to predict the values of a label on additional unlabeled data. Supervised learning is commonly used in implementations, where historical data shall predict future events [45].

Semisupervised Learning

Semisupervised learning resembles supervised learning to that extent, that both labeled and unlabeled data are utilized for training. Unlabeled data tends to be less expensive and takes less effort to acquire. This learning type can be used with methods such as classification, regression and prediction. It is especially useful when the cost associated with labeling is too high to allow for a fully labeled training process [45].

Unsupervised Learning

Another popular learning method is called unsupervised learning. It is used against data that has no labels. The algorithm is obliged to find the right answer by itself. The goal is to explore the data and find a structure within. This method of learning works well on transactional data, such as when it comes to identifying segments of customers with similar attributes, who can then be treated similarly in marketing campaigns. Other applications include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decomposition [45].

Reinforcement Learning

The fourth type of learning is reinforcement learning, which is often used in robotics, gaming or navigation. Through this particular learning method, an algorithm discovers through trial and error which actions yield the greatest rewards. The primary contents of this method consist of the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent is able to do). The objective for the agent is to choose actions maximizing the expected reward over a given amount of time. By following a good policy, the agent can accelerate finding the goal, which is in this case to learn the best policy [45].

3.2 Deep Learning

Deep learning can be seen as a subcategory between artificial intelligence and machine learning. It combines advances in computing power and special types of neural networks to learn complex patterns in large amounts of data successfully. Its techniques are state of the art for identifying objects in images and words in sounds. At the moment, researches tend to apply these successes in pattern recognition to more complicated tasks, such as automatic language translation, medical diagnoses and numerous other social and business problems [45].

3.3 Natural Language Processing

Natural language processing is the manipulation of the human language through a computer program. One of the most popular applications of NLP is the spam identification in mail communication, where the subject and the text of an email are investigated and classified into regular or junk mail. Its tasks are the translation of a text, the analysis of the sentiment in such and speech recognition [58].

3.3.1 Sentiment Analysis

Further, NLP works hand in hand with *Sentiment Analysis* (SA). Medium defines SA as the contextual mining of text, identifying and extracting subjective information in the source material, such as helping a business understanding the social sentiment of their brand, product or service by monitoring online conversations [47]. It is the automated process of understanding an opinion about a given subject from written or spoken language. It is also known as Opinion Mining. Besides identifying an opinion, these systems extract attributes of an expression, such as the polarity, the subject and the opinion holder. Polarity defines whether the opinion of a speaker expresses positivity or negativity. The subject is the object talked about, and the opinion holder is thus the person or entity expressing an opinion. Available information over the Internet is continually growing. Therefore a large amount of data expressing opinions can be gathered online. Unstructured information could be automatically transformed into structured data of public opinions with the aid of sentiment analysis. The resulting data can be beneficial for commercial usage like marketing analysis, public relations, and various other applications. Considering this fact, it seems to be a valuable method for detecting sarcasm

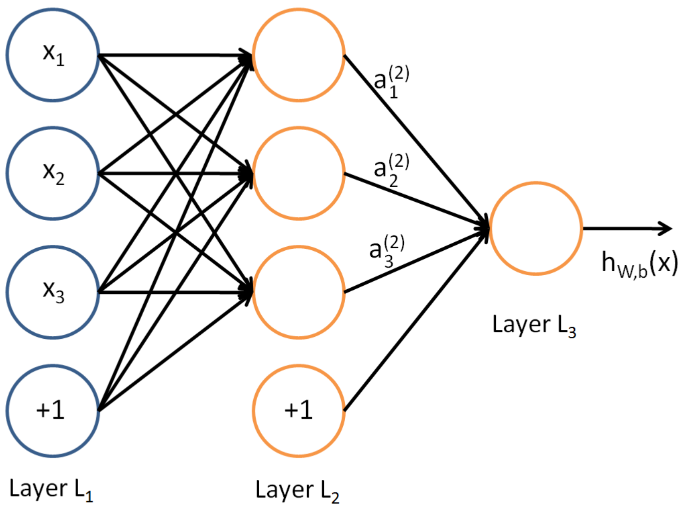


Figure 3.1: Illustration of a basic neural network structure [55]

in product reviews [52].

3.4 Deep Neural Networks

Neural networks have proven to outperform other algorithms in accuracy and speed as of yet. As the name already implies, neural networks consist of several neurons cohering. Those networks were inspired by the neural architecture of a human brain [55]. A basic structure of a neural network can be seen in the following image, figure 3.1. A neuron in the machine learning world is a placeholder for a mathematical function, and its only job is to provide output by applying the function on the inputs provided. The function used within a neuron is called an activation function. A layer in a neural network is nothing but a collection of neurons taking in input and providing output. Inputs to each of these neurons are processed through the activation functions assigned to the neurons. A neural network possessing more than one hidden layer is generally called a deep neural network [55]. The elements of deep neural networks, including the activation functions, are described in more detail in the following sections.

3.4.1 Activation Functions

The distinct activation functions provide a starting input for neural networks, determining the output a node will generate. The step function, the `sigmoid` function, the `tanh` function and the *Rectified Linear Unit* (ReLU) function belong to the most commonly used activation functions [55]. The most relevant for understanding the work done are described in chapter 4.

3.4.2 Convolutional Neural Network

CNNs is a form of neural network heavily used in computer vision. The hidden layers of a CNN are named convolutional layers, pooling layers, fully connected layers, and

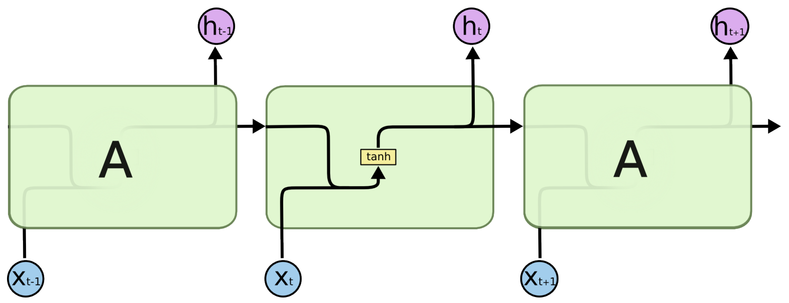


Figure 3.2: Illustration of an RNN structure [18]

normalization layers. Instead of using a simple activation function as input for this network, convolution and pooling functions are used. Convolution operates either on two signals (in a 1D space) or two images (in a 2D space), where one acts as an input signal and the other one (also called the kernel) as a filter on the input, producing an output. In simplified words, the input signal is multiplied with the kernel to get a modified signal. On the other side, pooling is a sample-based discretization process. The goal of pooling is to down-sample an input representation by reducing its dimensions and allowing for assumptions to be made about features in sub-regions. The two existing types of pooling are known as max and min pooling. Max pooling is based on picking the maximum value of a selected region. The opposite is described by min pooling, where the objective is picking the minimum value of a selected region.

All in all, CNNs are neural networks, consisting of hidden layers having convolution and pooling functions in addition to an activation function, aiming to introduce non-linearity [55]. To be more exact, the max-pooling layer takes the maximum of features over small blocks of a previous layer. The output indicates if a feature was present in a previous region, but not precisely where. Max-pooling layers allow later convolutional layers to work on larger sections of the data [24].

3.4.3 Recurrent Neural Network

RNNs are a form of neural network heavily used in the NLP area. In general neural networks, the input is processed, and an output is generated, without assuming that two successive input could be dependent on each other. This assumption is undoubtedly relevant in real-life scenarios. In the case that one wants to predict the next word in a sequence, dependence on previous observations must be considered. RNNs are called recurrent because they perform the same task for every element in a sequence. The output is thus dependent on previous computations [55]. Figure 3.2 shows the structure and behaviour of an RNN. RNNs reveal a chain-like structure, similar to a sequence or list. They already succeeded in solving problems such as speech recognition, language modeling, translation and image captioning [18]. RNNs can be thought of to be networks containing a memory, capturing information about the calculation process. Nonetheless, RNNs are limited to looking back only a few steps in practice. Regarding the structure of an RNN, one can imagine it as a neural network consisting of multiple layers, where each layer is representing the observations at a certain time. Those networks have shown

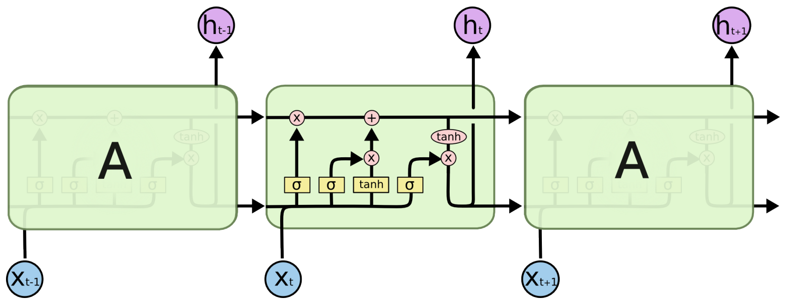


Figure 3.3: Illustration of an LSTM structure [18]

to be greatly successful in NLP, especially in their variant of an LSTM, which is able to look back longer than the usual RNN structure [55].

The standard RNN was developed by the idea of being able to connect previous information to the present task, such as a word in a sentence might indicate what the next word will most probably going to be. Considering a simple language model, a recent problem of standard RNNs can be shown. If a neural network is trying to predict the last word in the sentence “the clouds are in the sky”, there is no further context needed to be assured that the next word is going to be “sky”. Where the gap between relevant information and the place that a searched-for term is needed, is rather small, RNNs are able to learn past information. In cases, where more context is needed, such as in the sentence “I grew up in France... I speak fluent French.”, recent information suggests the next word to be the name of a language. If the network has to narrow down to which specific language is needed, more context is required. As a gap grows, RNNs become unable to memorize past information [18]. During backpropagation, those networks suffer from the vanishing gradient problem. Such gradients describe values used to update a neural network’s weights. The vanishing gradient problem occurs when the gradient shrinks as it back propagates through time. If a gradient value becomes too small, it does not contribute to the learning effect anymore [54]. The term LSTM describes one essential type of RNN. It is a particular kind of RNN, which works in an enhanced manner.

3.4.4 Long-Short Term Memory

LSTMs are specific RNNs, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber [18] in 1997 and since then refined and popularized by various researchers. Their default behaviour is to remember information for long periods. As already described, the typical structure of an RNN is a chain of repeating modules. In standard RNNs, the repeating module corresponds to a single `tanh` layer [18]. Figure 3.3 illustrates the functionality of an LSTM. Just like RNNs, LSTMs are

built of this chain-like structure. Other than RNNs, the repeating module consists of four neural network layers, denoted as a yellow box in the above diagram. The pink circles represent pointwise operations, such as vector additions. The core idea behind LSTMs is to obtain a particular cell state. It runs straight down the entire chain, having minor linear interactions. Thus it is facilitated for the information flow to remain unchanged. Individual gates are able to alter the information in a cell, composed of a `sigmoid` neural net layer and a pointwise multiplication operation [18].

To better describe how the information is flowing in an LSTM network, a more specific explanation is given. In the case of an online review, one would first look at the rating then determine if other persons thought a product was good or bad. While reading the review, the human brain subconsciously only remembers important keywords. An LSTM is able to learn to keep only relevant information to make predictions while forgetting about non-relevant data. Based on the words remembered, a decision about the sentiment in the review is made. LSTMs resemble the control flow of RNNs. They process data passing on information as it flows forward. Operations within the LSTM cell are used to either keep or forget information. The cell state transfers relevant information down the sequence chain. The state mimics the memory of the network. The cell state itself can carry relevant information to following time steps. During the state transfer, information is added or removed to the cell state via gates. Their goal is to decide which information is allowed on the cell state. This is achieved by calculating the `sigmoid` function, described further later on [54].

All in all, an LSTM consists of a forget gate, an input gate and an output gate. The forget gate decides which information shall be thrown away. The previous hidden state and current information are passed through the sigmoid function. The closer an output value is to 1, the likelier it will be passed on to the next gate. The input gate is utilized to update the cell state. The hidden state and the current input are also passed to the `tanh` function to help to regulate the network. The output of the `tanh` and the `sigmoid` function are multiplied, and the output will decide which information is essential. The cell state is pointwise multiplied by the forget vector. Afterwards, the input gate output is taken to do a pointwise addition updating the cell state to new values. Lastly, the output gate decides what the next hidden state will be. This state is specifically used for predictions. The previous hidden state and the current input are passed into the `sigmoid` function. The newly modified state is then passed to the `tanh` function. The `tanh` output is multiplied with the `sigmoid` output to decide upon the information that will be carried by the hidden state. The new cell state and the new hidden state are then carried over to the next time step, where the cycle starts over again [54].

3.4.5 Proposed Deep Learning Model

Ghosh et al. suggest a deep neural network, composed of an embedding layer, a two-layer CNN, an optional dropout layer, a two-layer LSTM network, a dense layer and a softmax layer, for computing any text sentiment detection task. A rough overview can be seen in figure 3.4. Neural network architectures, such as CNNs, RNNs, and others have shown excellent capabilities for modelling complex word composition in a sentence. A sarcastic text can be considered elementally as a sequence of text signals or word combinations. An LSTM has the capability to remember long distance temporal dependencies. Moreover, as it performs temporal text modelling over input features, higher-level modelling can distinguish factors of linguistic variation within the input. CNNs can also capture temporal text sequence through convolutional filters, which connect a subset of the feature space that is shared across the entire input. They also

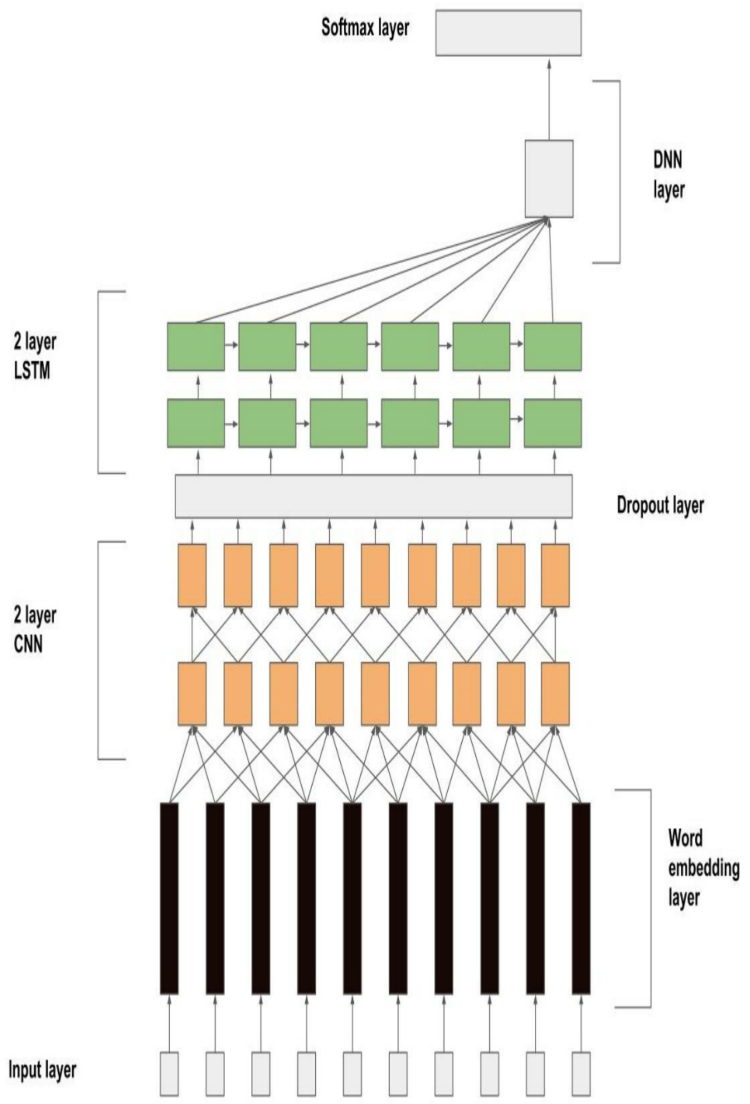


Figure 3.4: Illustration of the proposed DNN structure by Ghosh et al. [14]

reduce frequency variation and can directly capture temporal text patterns for shorter texts. However, in longer texts, where temporal text patterns may span across 15 to 20 words, CNNs must rely on higher-level fully connected layers to model long-distance dependencies. This dependency occurs because the maximum convolutional filter width for a text is 5. Another major limitation of CNNs is the fixed convolutional filter width, which is not suitable for different lengths of text patterns and cannot always resolve dependencies properly. Regarding CNNs, obtaining the optimal filter size is expensive and corpus dependent, while the LSTM operates without a fixed context window size. Its performance can be improved by providing better features [14].

Following the proposal of Vincent et al. [40], it can be beneficial to exploit a CNN's ability to reduce frequency variation and map the input features into robust composite features for using them as an input to an LSTM network. After that, dense layers seem to be appropriate for mapping features into a more separable space. A fully connected dense layer, added on top of an LSTM network, can provide better classification by mapping between output and hidden variables respectively by transforming features into an output space.

3.5 Understanding Sarcasm

Sarcasm is a linguistic construct in conversations developed by the human race. The understanding of such a construct is thus dependent on each person itself.

3.5.1 Definition

According to Wicana et al. [41], irony is the use of words to express the opposite of the literal meaning of a sentence, while sarcasm is defined as having a “bitter, caustic” tone directed against an individual. When a person is trying to be sarcastic, he or she is communicating an ostensible message to their opposite, but simultaneously framing the message with a metamessage, saying something like “I don’t mean what I just said, I meant the exact opposite” [17].

3.5.2 Psychological Aspects

The Smithsonian Magazine states that sarcasm often seems to exercise the brain more than sincere statements do. Scientists have monitored the electrical activity of the brain when exposed to sarcastic statements. They have investigated that brains have to work harder to understand sarcasm. Learning to understand sarcasm includes developing a “theory of mind” to see beyond the literal meaning of the words and understand that the speaker may be thinking of something entirely different.

Many parts of the brain are involved in processing sarcasm, including the temporal lobes and the parahippocampus. Both are involved in picking up the sarcastic tone of voice. Researches expose that the left hemisphere of the brain seems to be responsible for interpreting literal statements. On the other hand, the right hemisphere and both frontal lobes seem to be involved in figuring out when an utterance is intended to mean exactly the opposite [8].

3.5.3 Typical Features

The tone of the voice has proven to be a strong indication of sarcasm. Speakers often use a nasal tone when speaking in a sarcastic manner. This is because they immediately link sarcasm to disgust. The other manner a sarcastic person uses is called inverse pitch obtrusion. In other words, a particular word gets stressed at a lower pitch.

Imagining the following example, the process can be more easily understood. The stress on the word “great” varies depending on whether the speaker acts sarcastic or sincere: When saying the sentence “Great weather, huh?”, a high pitch implies sincerity, while a low pitch implies sarcasm usage. Other implications of sarcasm would be the elongation of words and saying words associated with excitement in a flat tone. The facial expression also matters when signifying sarcasm. This is why it is oftentimes so hard to detect in text. Text cannot convey tone and must rely only on context. Despite the context, many people cannot understand sarcasm, as a fact of certain parts of their brain being damaged or unable to recognize sarcastic statements [37].

3.5.4 Human Capacity

Researchers have found that kids pick up the ability to detect sarcasm at a young age. They showed children short puppet shows in which one of the puppets made either a literal or a sarcastic statement. The children had thus decided whether the puppet was friendly or mean. Children as young as 5 were able to detect sarcastic statements quickly [8].

Autistic people may have trouble understanding sarcasm, because they struggle to connect context, intention and language, but interpret speech literally. Not being able to understand sarcasm can also be a sign of dementia. Deterioration in certain parts of the brain corresponded with an inability to detect lies (or sarcasm). This finding may help doctors diagnose dementia earlier. Therefore a patient could receive treatment sooner [37].

3.5.5 Limits of Machine Detection

The two biggest, historical (and ongoing) problems in machine learning appear to be overfitting (in which the model exhibits bias towards the training data and does not generalize to new data, meaning that it learns random things when trained on new data) and dimensionality (algorithms with more features work in higher or multiple dimensions, making understanding data more difficult). One of the most common mistakes among machine learning beginners is testing training data successfully and having the illusion of success. The importance of keeping some of the data set separate when testing models, and only using that reserved data to test a chosen model, is significantly emphasized. When a learning algorithm is not working, often the quickest path to success is to feed the machine more data. However, feeding more data can lead to issues with scalability, in which we have more data available but less time to learn that data, which remains an issue. In terms of purpose, machine learning is not an end or a solution in itself [43].

Chapter 4

Experimental Evaluation

Evaluation of the work takes place after the first results were output by the ML model. The project work itself consisted of building a model to detect sarcasm in product reviews, achieved by implementing a logistic regression model on the one hand. The second model implemented was a deep learning structural model, where the results should be compared to those of the standard logistic regression one. The reason for utilizing a more deeply structured algorithm had to be emphasized, and the usefulness of employing it had to be proved. Before diving into the results and the details of the evaluation, the project's structure and architecture are emphasized.

4.1 Architecture

The experimental approach was conducted on the latest version of *MacOS*, which is MacOS Mojave[1]. The version of the utilized browser regarding the project architecture is *Chrome*[2]. The version of the web environment is crucial, since the project code was implemented and stored on the cloud. The development environment used is further described in section 4.1.1.

4.1.1 Google Colaboratory and Jupyter

Google Colaboratory is a cloud service based on *Jupyter Notebooks.* It provides free-of-charge access to a robust GPU, as well as a runtime fully configured for training deep learning networks [6]. Before introducing Google Colaboratory, the technology incorporated within it should be discussed. This technology is called Jupyter Notebooks. Jupyter is an open-source, browser-based tool integrating interpreted languages, libraries, and tools for visualization [33]. Each Jupyter document is composed of multiple cells, that can be stored locally or on the cloud. Such a cell contains script language or markdown code. The output of each cell, which can be run individually, is embedded in the document. Typical outputs include text, tables, charts, and graphics [35].

Google Colaboratory combines the possibilities of writing and executing code in the same *User Interface* (UI), while instantly showing every kind of change in the execu-

[1]version 10.14.6
[2]version 76.0.3809.132

tion results after altering any minor part of available code sections. Colaboratory allows combining code, comments, multimedia, and visualizations in an interactive document, which can be shared, re-used, and re-worked. Colaboratory works with most major browsers and is tested with the latest versions of Chrome, Firefox and Safari. Colaboratory is entirely free of charge. The main difference to Jupyter, the open-source project on which Colaboratory is based, is, that Colaboratory allows its users to use and share Jupyter notebooks with others without having to download, install, or run anything on their own computer other than a browser. The Jupyter notebooks can be shared via Google Drive with any desired group of people. Regarding language support, Colaboratory supports Python 2.7 and Python 3.6, preconfigured with the essential machine learning and artificial intelligence libraries, such as TensorFlow, Matplotlib, and Keras. When the code in the notebook is run, it is executed in a virtual machine dedicated to one's account. After being idle a certain amount of time, the virtual machine is recycled, due to the maximum lifetime of each machine enforced by the system [46]. Other than that, Google Colaboratory provides a solid base for creating a runtime efficient deep learning model, accelerated by GPU, and hosted on the Google Cloud platform.

Carneiro et al. [6] provide a study about the performance analysis of Google Colaboratory as a tool for accelerating deep learning applications. Due to their findings, Colaboratory was chosen for implementing the project's deep neural network. Their objective was to verify whether it is advantageous using Colaboratory for processing modern deep learning applications rather than dedicated hardware. Their research states that using Colaboratory's accelerated runtime for processing the deep learning applications and the *Graphics Processing Unit* (GPU)-centric combinatorial search is faster than using 20 physical *Central Processing Unit* (CPU) cores. Therefore, it is worth using than, for instance, a robust server with no GPU, a laptop, or a workstation that needs to be configured and has a mid-end GPU. Nevertheless, the free-of-charge hardware resources provided by Colaboratory are not enough to solve demanding real-world problems, but it is indeed enough for research reasons. For this reason, Google Colaboratory was chosen over any conventional *Integrated Development Environment* (IDE), such as PyCharm or Eclipse.

4.1.2 Python

According to *TutorialsPoint*, *Python* is a high-level, interpreted, interactive and object-oriented scripting language. The language is designed to be highly readable, meaning it uses English keywords frequently whereas other languages use punctuation. All in all it has fewer syntactical constructions than other languages. Python is especially present in the Web Development Domain and described as a must for specialists working in this field. Python is a language processed at runtime by the interpreter. Therefore there is no previous compilation of the code obligatory. This is similar to the coding languages Perl and *PHP: Hypertext Preprocessor* (PHP). Python applies coding interaction, where a user can sit in front of the Python prompt and directly interact with the interpreter. Furthermore, the language supports *Object-Oriented Programming* (OOP), allowing to encapsulate program code within objects. Lastly, it is a beginner's language, easy to learn, mainly due to the full range of applications from simple text processing to game architecture design [61].

Python, Python 3 specifically, is the most recent and popular coding language for neural network and modern machine learning applications. It has a wide range of available libraries and frameworks for implementing AI software, many of these tackling the different layers of neural network creation, such as *TensorFlow*, *Keras* and *Scikit-Learn*. In combination with those libraries, the choice for a fitting programming language and version was fixed on Python 3[3]. It is currently the most updated and likewise stable version on the market.

4.1.3 Keras and TensorFlow

Keras is a so-called *Application Program Interface* (API), written in Python and capable of running on top of the libraries TensorFlow, *Microsoft Cognitive Toolkit* (CNTK), or *Theano*. Keras is particularly designed for deep learning problematics and allows for simplified and accelerated prototyping through user-friendliness, modularity, and extensibility. The interface support CNNs, as well as RNNs, and combinations of both. Additionally, it seamlessly runs on CPU and GPU. Keras follows best practices for reducing cognitive load, by offering consistent and simple APIs, minimizing the number of user actions required for common use cases, and providing clear feedback upon user error. In Keras, neural networks, cost functions, activation functions, optimizers and regularization schemes are all standalone modules, creating a new model when combined [21].

As a backend engine, the engine TensorFlow was installed. TensorFlow is an end-to-end open-source platform for machine learning. It allows easy building and training of machine learning models using intuitive high-levels APIs like Keras. Python development is possible on CPU, GPU and *Tensor Processing Unit* (TPU). TPUs describe a chip technology, particularly created for TensorFlow by Google, to accelerate the process of machine learning. TensorFlow's high-level APIs are based on the Keras API standard for defining and training neural networks. Keras, therefore, enables fast prototyping, state-of-the-art research, and production [60]. Minding the installed Python 3 version, being the latest version on the market, Keras's latest version[4], and TensorFlow's supported version[5], were installed.

4.1.4 Scikit-learn

Scikit-learn is described as the gold standard of ML. Additionally, it provides a wide selection of supervised and unsupervised learning algorithms. Similar to those mentioned above, its core API design revolves around being easy to use, powerful, and still maintaining flexibility for research endeavours. Scikit-learn is built on top of several common data and math Python libraries, simplifying the process of integrating between them all. Scikit-learn offers implementations of the libraries *NumPy*, *SciPy*, *Matplotlib*, *IPython*, *Sympy*, and *Pandas*. They all offer different application foci, such as matrix operations, scientific computing, data visualizations, data manipulation, or data analysis. The framework's robust set of machine learning applications include Regression

[3]version 3.7.2
[4]version 2.2.5
[5]version 1.14.0

for fitting linear and non-linear models, Clustering for unsupervised classification, Decision Trees for both classification and regression tasks, Neural Networks for end-to-end training also in both tasks, SVMs for learning decision boundaries, and Naive Bayes for probabilistic modelling. In contrast to other existing libraries, Scikit-learn provides feature manipulation methods, outlier detection and model validation [59]. The two most interesting methods for this project implementation, therefore, are cross-validation and hyperparameter tuning.

Regarding the already used versions of TensorFlow and others, Scikit-learn's latest version[6], was chosen for programming. Instead of providing as many features as possible, the project's goal has been to provide solid implementations. Scikit-learn offers a reliable implementation for creating an efficient ML application.

4.1.5 Natural Language Toolkit

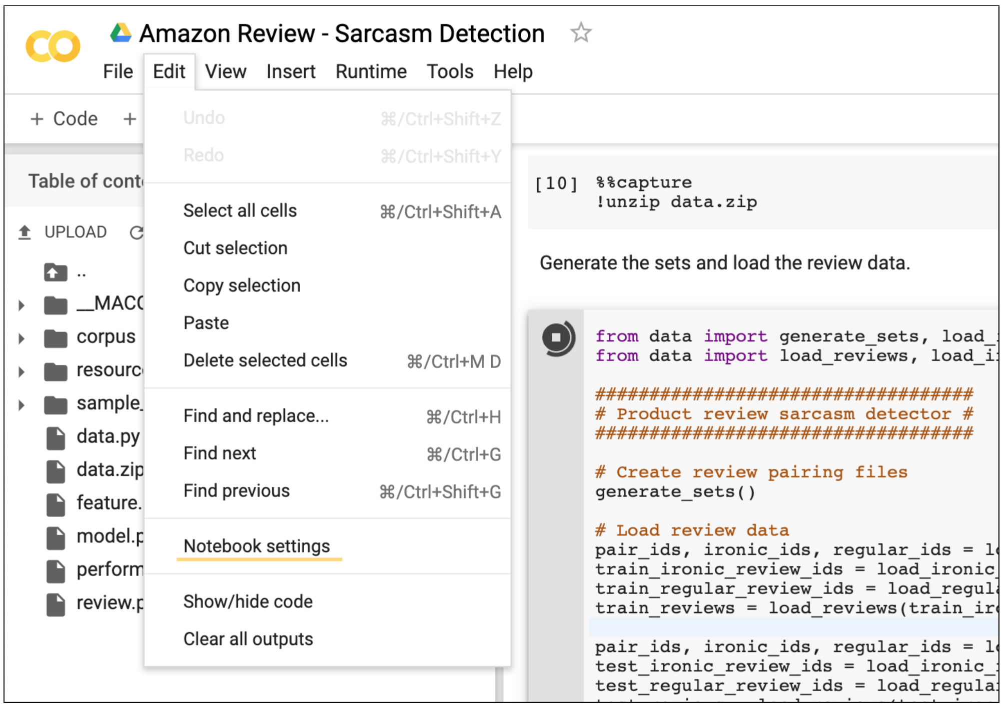
The *Natural Language Toolkit* (NLTK) is the leading platform for building Python programs to work with human language data. The framework provides a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Thanks to the comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, as well as researchers. The framework is a free, open-source, community-driven project [4]. NLTK offers the most important preprocessing methods for data cleaning and further neural network training. In the case of the current research, the most valuable methods, for this reason, are *Possessive ending parent's* (POS) tagging, and sentence and word tokenization.

4.1.6 Gensim and Word2vec

Gensim is defined as a Python library utilized for topic modelling, document indexing and similarity retrieval with large corpora. One of the target audiences is the NLP community. All its algorithms are memory-independent, which means it can process input that is larger than the *Random Access Memory* (RAM) available. The library also provides efficient multicore implementations of popular algorithms, such as online *Latent Semantic Analysis* (LSA), *Latent Dirichlet Allocation* (LDA), *Particle give up* (RP), *Hierarchical Dirichlet Process* (HDP) or Word2vec deep learning. Gensim can run LSA and LDA on a cluster of computers [36]. Its easy integration made it possible to incorporate the library and its application of Word2vec rapidly.

Word2vec contains a group of related models producing word embeddings. The models provided are shallow, two-layer neural networks trained to reconstruct linguistic contexts of words. Word2vec takes a large corpus of text as an input and produces a vector space with each unique word in the corpus being assigned to a corresponding vector in the space. Word vectors are positioned in a way, that words that share common contexts in the corpus are located close to one another in the vector space [30]. The process of training such a model is further explained in section 4.2.

[6] version 0.21.3



```
[10] %%capture
     !unzip data.zip
```

Generate the sets and load the review data.

```
from data import generate_sets, load_
from data import load_reviews, load_i

####################################
# Product review sarcasm detector #
####################################

# Create review pairing files
generate_sets()

# Load review data
pair_ids, ironic_ids, regular_ids = l
train_ironic_review_ids = load_ironic
train_regular_review_ids = load_regul
train_reviews = load_reviews(train_ir

pair_ids, ironic_ids, regular_ids = l
test_ironic_review_ids = load_ironic_
test_regular_review_ids = load_regula
```

Figure 4.1: Extract of Google Colab's user interface

4.2 Implementation

The implementation of the project's source code takes place on Google Colaboratory, within a Jupyter notebook. Regarding the coding style, the project's needs strived for a combination of modules and inline, notebook coding. Therefore, an adequate level of encapsulation was preserved, while interactivity of the user within the notebook is not ceased. Coding in a Jupyter notebook offers the programmer to visualize results fast and easy. These were the major reasons why this platform was chosen for the project's use case.

4.2.1 Configuration and Usage

In Google Colaboratory, the user has the possibility to program using either Python 2 or Python 3. The settings can be adjusted in the Colab specific notebook settings, as can be seen in figure 4.1. The project's configuration needed to be Python 3. These were all the settings configured in this case. The needed imports for the project implementation are displayed below.

```
!pip install q numpy==1.16.4
!pip install q matplotlib==3.1.1
!pip install q tensorflow==1.14.0
!pip install q scikit-learn==0.21.3
!pip install q keras==2.2.5
!pip install q gensim==3.8.0
```

```
!pip install q nltk==3.4.5
```

Numpy is used for several matrix operations within the program, while Matplotlib is needed for the learning curve, accuracy and loss visualization. TensorFlow, Scikit-learn and Keras are the main libraries commonly utilized in DL applications. Gensim is an open-source toolkit for unsupervised topic modelling and NLP, using modern statistical ML. This toolkit offers Word2vec, a group of related models that are used to produce word embeddings. Lastly, NLTK provides a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Within the program, it is imported for sentence and word tokenization prior to training the neural network.

```
import nltk
nltk.download('punkt')
nltk.download('maxent_treebank_pos_tagger')
```

The above-defined code section describes some obligatory NLTK downloads to ensure that the used tokenizers within the program implementation work as intended. A last configurational edit is made through the following code lines.

```
import os

if not os.path.exists('./model/'):
  os.makedirs('./model/')

if not os.path.exists('./src/'):
  os.makedirs('./src/')

if not os.path.exists('./images/'):
  os.makedirs('./images/')
```

The `os` library is used to achieve a structured directory tree. The model directory contains all the cached models of neural network training. The source folder comprises the project's Python classes and functions. In the image folder, the accuracy and loss plots, as well as other illustrations, are stored. The whole directory tree is illustrated in figure 4.2 below. The corpus directory contains all the review entries in differently stored formats, further described in section 4.2.2. The resources folder is part of the data preprocessing procedure. The files within, `negative-words.txt` and `positive-words.txt`, are collections of commonly known, either negatively or positively rated words structured as a list.

4.2.2 Data Fetching

As stated in the sections 1.2.1 and 2.3, the corpus used for evaluating the deep neural network is the product review data set by Elena Filatova, containing entries labeled as sarcastic or non-sarcastic. The data set is loaded in the following manner. The actual *Hypertext Markup Language* (HTML) review files are stored in the directory called

```
corpus/
├── Ironic/
│   ├── *.html
│   └── *.txt
├── Regular/
│   ├── *.html
│   └── *.txt
├── five_labels_plus_stars.xlsx
└── file_pairing.txt
images/
└── *.png
model/
└── *.hdf5
resources/
├── negative-words.txt
└── positive-words.txt
src/
├── data.py
├── feature.py
├── model.py
├── performance.py
└── review.py
```

Figure 4.2: Sarcasm detection directory tree

"corpus". Within this directory, they are split into two different folders, called "Ironic" and "Regular". The naming convention relies on an individual, unique product *Identification* (ID), for instance `1_1_R280644F3NWFFN.html`. Every HTML file is available in a text file within the same folder, using the identical ID in the file name, besides the file ending being `.txt`. Therefore, ironic and neutral product reviews are sorted within these two separate folders.

Additionally, a text file named `file_pairing.txt` is defined in this folder structure. This file describes the order of the review content. For some of the product reviews, a pair of files exists, meaning that for the same product, one sarcastic review, as well as a regular review was collected. These reviews are labeled as "PAIR" within the file. Singular reviews are either labeled as "IRONIC" or "REGULAR", following the ID of the file. This labeling is crucial for data set loading, loading the product reviews in particular. The file `file_pairing.xlsx` is provided as an addition to the data set by Filatova, but is not further used in the realization of the project and therefore discarded in the following steps.

Data structure

First of all, the data entries are analyzed according to their structural components. Elena Filatova [12] already visualized the distribution of the product reviews in her research, as can be seen in table 4.1. The total collection of data entries consists of

		Number of reviews with				
		1*	2*	3*	4*	5*
sarcastic	437	262	27	20	14	114
regular	817	64	17	35	96	605

Table 4.1: Number of data entries and their distribution amongst the star rating according to their label [12]

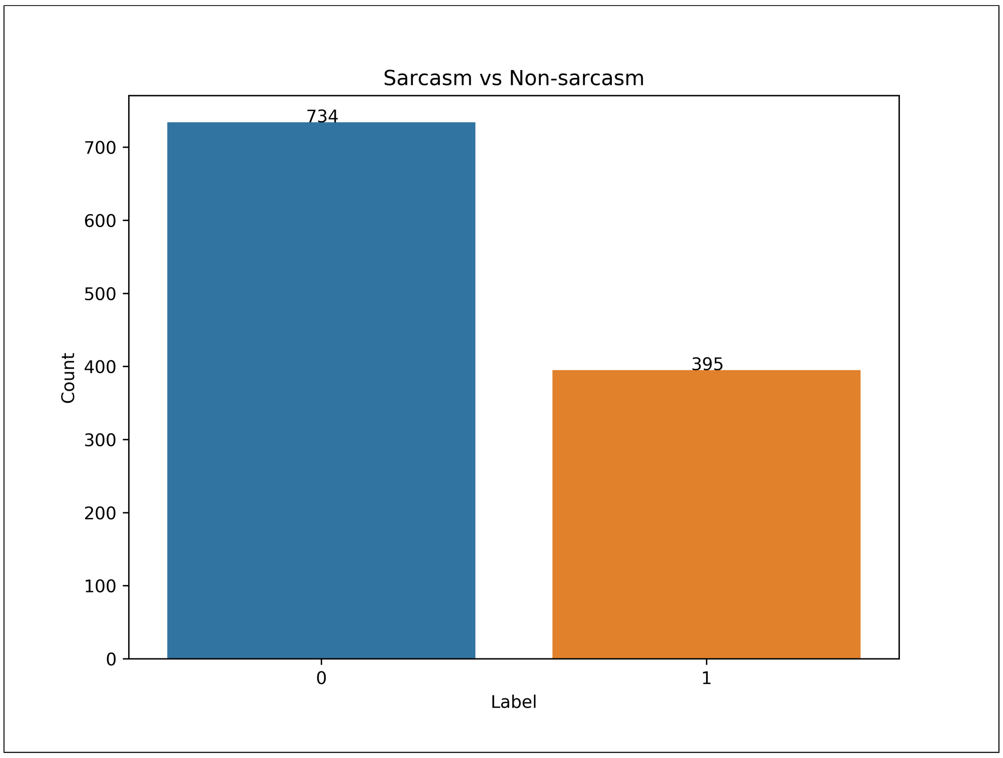


Figure 4.3: Distribution of the product reviews in the training set

437 sarcastic and 817 regularly tempered product reviews. In the table, there is a clear tendency of one-star rated products being reviewed sarcastically. As opposed to this, five-star rated product tend to be reviewed neutrally.

The following illustrations in figure 4.3 display the distribution of product reviews explicitly for the training data set used when training the neural network.

The distribution graphs confirm that 90 percent of the whole product review collection is used for neural network training. The horizontal axis defines two variables. Variable 0 characterizes the neutral tempered product ratings, variable 1 defines the sarcastic ones. Figure 4.4 shows the distribution of the number of words used in the product review descriptions. The average number of words in a product review is approximately 242 words. The minimum number of a typical review is 13 words. The maximum number of words is 3571. Filatova aimed to collect well-written reviews, poorly written review texts

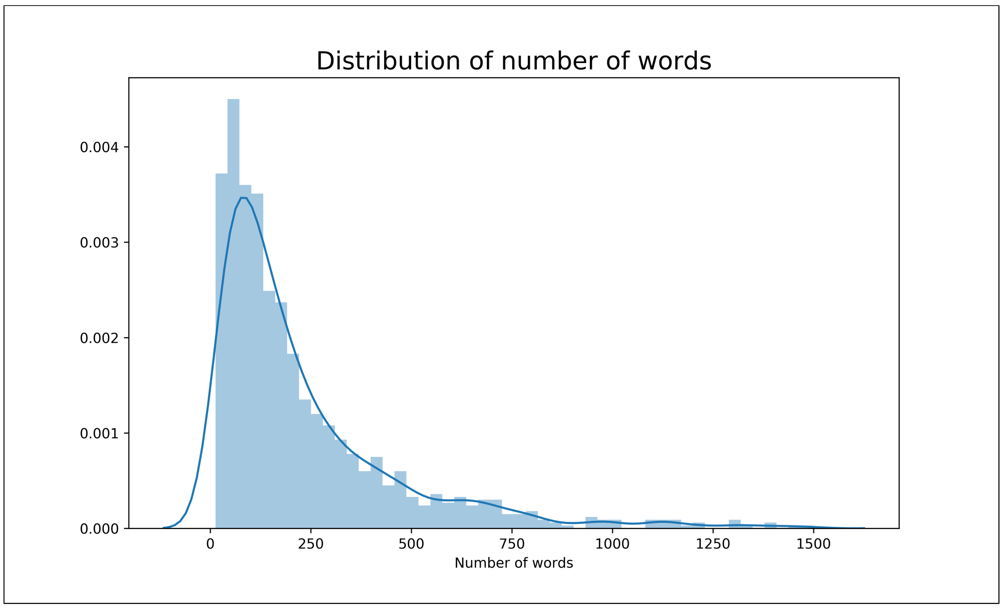


Figure 4.4: Distribution of the words in the product review texts

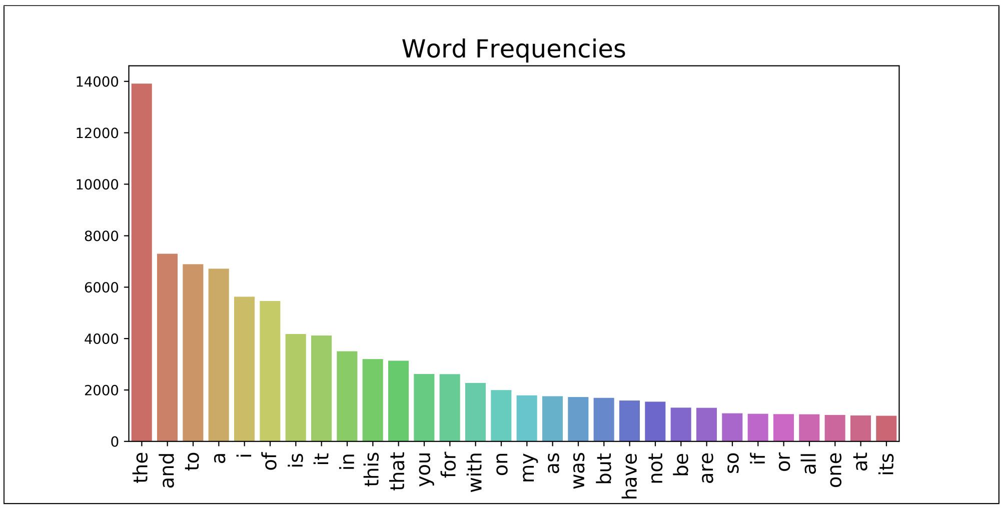


Figure 4.5: Frequency of the most used words in the product review texts

were discarded anyhow. that is why reviews having less than ten words do not exist in the data set. Figure 4.5 illustrates the occurrence of the most used terms within the data set. The overall majority of the words being used most of the time are considered stop words. Stop words are terms that have little meaning for training a neural network. In this particular case, stop word removal is not performed, as these words do have a major influence on the context of a sentence. The word "not" would be considered as a stop word in a sentence, but it could make a significant difference regarding the comprehension of the sentence's meaning. The preprocessing is further described in section 4.2.3.

Data loading into Colab

The data loading process is simple in Google Colaboratory, nevertheless, it requires a few tricks to function correctly. Using the Colab specific library `google.colab`, the tool offering the opportunity to upload a zipped file is imported. When running a code section in Colab, also called a cell, the output is immediately printed below this cell. The output in case of the file upload, therefore, is a dialog, where a specific file can be chosen from any local directory. In this case, the file was the zipped file `data.zip`, comprising the corpus and the resources directory.

```
from google.colab import files
uploaded = files.upload()
```

The next step is to unzip the data folder to continue operating on the neural network using it. The command `%%capture` is one of the predefined IPython magic commands, and has the effect of suppressing the cell output, only giving information about every individual unzipped file in this particular case.

```
%%capture
!unzip data.zip
```

The content of the zipped file comprises the ironic and regular ID files, as well as the file pairing text file and the distribution table. The data unpacked is now ready to be utilized within the Jupyter notebook.

Data loading within the program

In the program code, minor adjustments have to be made, to simplify the reading of the review files for further data processing. The function `generate_sets()` contains the functionality for doing so. There, the function `create_data(sizes)` is called, where one can decide on the splitting of training, test and validation data. To have a balanced split of data sets, the training set was defined to have 90 percent of the complete data, while the test set was determined to have only 10 percent. As there were only few review entries to train a DNN, the training set is defined to provide a high number of reviews, to allow the neural network to be successfully trained on a rather high number of reviews and therefore output desirable results. Yet, a small amount of test data has to be provided for validating the functionality of the network.

```
# Create review pairing files
generate_sets()
```

To begin with fetching the data entries, the function `read_ids(filename)` is called, where the different IDs in the file `file_pairing.txt` are sorted into related lists. The following section gives an impression of how the review IDs are ordered in the text file.

```
PAIR:  34_6_R2WH104QVDL305 (ironic) 34_6_R2PCIAGT8AVU67 (regular)
IRONIC:  14_1_R1MEA9N3D7643N
REGULAR: 18_7_R34LXBIHMBM1YY
```

Using the following program code, these lines can be read in and appropriate, filled lists are returned by the function. Using `codecs.open()` the file is read line by line into a variable. The read-in review IDs are then sorted into corresponding lists, either describing pairs, individual ironic or individual regular review entries.

```
# 'utf-8-sig' removes leading Byte Order Mark
with codecs.open(filename, 'r', encoding=fileEncoding) as ids_file:
    # Read lines of review pairing file
    review_raw = ids_file.readlines()

# Organize entries regarding labeling
for raw_id in review_raw:
    id_parts = raw_id.split()
    if id_parts[0] == "PAIR:":
        review_pair.append((id_parts[1], id_parts[3]))
    elif id_parts[0] == "IRONIC:":
        review_ironic.append(id_parts[1])
    elif id_parts[0] == "REGULAR:":
        review_regular.append(id_parts[1])
```

All the review IDs returned by the function are then added up to one large, labeled string list. The function `label_set()` is just returning a list of ID, label pairs, while `set_to_string()` returns a list of string representations for every ID, label pair.

```
# Add up labeled review ids to one large list
review_ids = set_to_string(label_set(review_ironic, "ironic"))
review_ids += set_to_string(label_set(review_regular, "regular"))
review_ids += set_to_string(label_set([r for i,r in review_pairs], "regular"))
review_ids += set_to_string(label_set([i for i,r in review_pairs], "ironic"))
```

A further step involves calling the function `divide_data(review_ids, sizes)`, dividing the data according to their percentages and return a list of lists.

```
def divide_data(data, sizes):
    # Return a list of list from the data accordingly to the given percentages
    assert not sum(sizes) > 100

    # Randomly shuffle data using given seed
    random.seed(44)
    random.shuffle(data)

    result = []
    num_sets = 0
    offset = 0

    # Check for zero in set size
    for elem in sizes:
        if elem != 0:
            num_sets += 1

    # Divide the entries within list
    for num in range(num_sets):
        if num == num_sets-1:
            result.append(data[offset:])
        else:
            end = offset + round(len(data) * sizes[num] / 100)
            result.append(data[offset:end])
            offset = end

    return result
```

At the beginning of the code section, the percentage distribution is checked on correctness, while right after that, the data set is shuffled using `random.shuffle(data)`. Then, the data are split into the given percentages, which were 90 percent of the data for training and 10 percent for testing. The data sections are appended to a list and returned. Finally, the generated sets are stored in individual files, called `training_set.txt` and `test_set.txt`. The following cell defines the loading of the training and test data into local Python variables, stored on Google Colaboratory. To load the training-specific IDs, `load_ids()` is called.

```
# Load review data
pair_ids, ironic_ids, regular_ids = load_ids("training_set.txt")
train_ironic_review_ids = load_ironic_review_ids(pair_ids, ironic_ids)
train_regular_review_ids = load_regular_review_ids(pair_ids, regular_ids)
train_reviews = load_reviews(train_ironic_review_ids, train_regular_review_ids)

pair_ids, ironic_ids, regular_ids = load_ids("test_set.txt")
test_ironic_review_ids = load_ironic_review_ids(pair_ids, ironic_ids)
test_regular_review_ids = load_regular_review_ids(pair_ids, regular_ids)
test_reviews = load_reviews(test_ironic_review_ids, test_regular_review_ids)
```

The function `load_ids(filename)` stores all review IDs into the corresponding list. The different lists holding review pairs, sarcastic reviews and regular reviews are, therefore, returned.

```
def load_ids(filename):
    # Load all review ids into lists
    if filename == None:
        print("Filename none")

    pair_ids = []
    ironic_ids = []
    regular_ids = []

    # 'utf-8-sig' removes leading Byte Order Mark (BOM)
    with codecs.open("./corpus/" + filename, 'r', encoding='utf-8-sig') as idsFile:
        rawReviewIDs = idsFile.readlines()
    for rawID in rawReviewIDs:
        IDParts = rawID.split()

        if IDParts[0] == "PAIR:":
            pair_ids.append((IDParts[1], IDParts[3]))
        elif IDParts[0] == "IRONIC:":
            ironic_ids.append(IDParts[1])
        elif IDParts[0] == "REGULAR:":
            regular_ids.append(IDParts[1])

    return pair_ids, ironic_ids, regular_ids
```

The function `load_ironic_review_ids(pair_ids, ironic_ids)`, as well as the function `load_regular_review_ids(pair_ids, regular_ids)`, splits the paired IDs, sorts them and adds them either to the ironic IDs list, or the regular IDs list, and returns each of these lists at last.

```
def load_reviews(ironic_ids, regular_ids):
    # Load all reviews into a dictionary
    review_dict = {}
    review_dict.update(read_reviews(ironic_ids, "./corpus/Ironic/", ironic=True))
    review_dict.update(read_reviews(regular_ids, "./corpus/Regular/", ironic=False))
    return review_dict
```

The function `load_reviews(ironic_ids, regular_ids)` loads all reviews, in this case, either the reviews for the training or the ones for testing, into a dictionary. This is done by reading the data review by review.

```
def read_reviews(review_ids, folder, ironic):
    # Return a dictionary containing reviews to the given IDs
    return {review_id: Review(filename="{0}{1}.txt".format(folder, review_id), ironic=ironic)
            for review_id in review_ids}
```

As opposed, the above function returns a dictionary containing the IDs pointing to review objects. The program classes needed for understanding are later described in section 4.2.2. Also, the data preprocessing happens there.

```
# Combine review ids
train_review_ids = train_ironic_review_ids + train_regular_review_ids
test_review_ids = test_ironic_review_ids + test_regular_review_ids
```

Lastly, the IDs of all training and testing specific review data entries are combined. This particular cell has to be run only once for initialization purposes. Summarizing, this cell creates the two files used for loading the review-specific data within the ML program.

Program classes

The project implementation comprises exactly three classes defining a single object for review display. All the classes to be described are located in the module `review.py`. The first class `Review` is illustrated and further explained below.

```
class Review(object):

    HTMLParser = HTMLParser()
    sentenceTokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
    wordTokenizer = TreebankWordTokenizer()

    def __init__(self, rawReview=None, filename=None, ironic=None):
        if filename is not None:
            self.parse_file(filename, ironic)
        elif rawReview is not None:
            self.parse(rawReview, ironic)
        else:
            self.product = ""
            self.title = ""
            self.text = ""
            self.stars = 0.0
            self.author = ""
            self.date = datetime()
            self.ironic = None
            self.wordSpans = []
            self.wordPolarity = []
            self.sentenceSpans = []
```

The class implementation starts with defining an HTML parser. For class internal word tokenization of the review text, the function `TreebankWordTokenizer()` is used, the function invoked by the NLTK library function `word_tokenize()`. While initializing the class object, usually the first class method called is `parse_file(self, filename, ironic, fileEncoding='latin-1')`.

```
    def parse_file(self, filename, ironic, fileEncoding='latin-1'):
        with codecs.open(filename, 'r', encoding=fileEncoding) as reviewFile:
            self.parse(reviewFile.read(), ironic)

    def parse(self, rawReview, ironic):
        self.ironic = ironic
        self.stars = float(get_subpart(rawReview, "<STARS>", "</STARS>"))
        self.author = get_subpart(rawReview, "<AUTHOR>", "</AUTHOR>")
        self.product = get_subpart(rawReview, "<PRODUCT>", "</PRODUCT>")

        # Date can be written in one of three formats
        try:
            self.date = datetime.strptime(
                                get_subpart(rawReview, "<DATE>", "</DATE>"),
                                "%B %d, %Y")
        except ValueError:
            try: self.date = datetime.strptime(
                                get_subpart(rawReview, "<DATE>", "</DATE>"),
                                "%d %B %Y")
            except ValueError:
                self.date = datetime.strptime(
                                get_subpart(rawReview, "<DATE>", "</DATE>"),
                                "%d %b %Y")
        self.title = get_subpart(rawReview, "<TITLE>", "</TITLE>")

        self.text = get_subpart(rawReview, "<REVIEW>", "</REVIEW>").strip()
        self.text = self.preprocess(self.text)
        self.sentences = self.tokenize_sentences(self.text)
```

Parsing the file starts by passing every raw review to the parsing function `parse(self, rawReview, ironic)`. There, every section of the HTML raw review text is returned by the method `get_subpart()`. Using the HTML-specific highlight words, such as "<STARS>" and "</STARS>", as indices for the review array, every subtext is extracted. In the case of the `star` variable, it is the integer number of stars of the product rating. The method is applied to every HTML subarea. The two last functions called are described in detail in the following code section.

```
    def preprocess(self, text):
        # Preprocess the given text by unescaping HTML entities
        return self.HTMLParser.unescape(text)

    def tokenize_sentences(self, text):
        return [Sentence(text)
                for text in self.sentenceTokenizer.tokenize(self.text,
    realign_boundaries=True)]
```

Preprocessing the given text just involves unescaping the HTML entities. Unescaping means to convert HTML to a common Python string. As the text is converted to a common string now, it serves as an input to the function `tokenize_sentences(self, text)`. There, a vector of `Sentence` objects is returned.

```
class Sentence(object):

    POS_TAGGER = 'taggers/maxent_treebank_pos_tagger/english.pickle'
    wordTagger = nltk.data.load(POS_TAGGER)
    wordTokenizer = TreebankWordTokenizer()
    __slots__ = ['text', 'words']

    def __init__(self, text):
        start = None
        end = None
        self.text = text
        self.words = self.tokenize_words(self.text)
```

The class uses a POS tagging mechanism to tag every single word in a sentence. The review sentence again has to be tokenized into words, additionally tagged at the same time, in the function `tokenize_words(self, text)`.

```
    def tokenize_words(self, text):
        return [Token(w,p)
               for w,p in self.wordTagger.tag(self.wordTokenizer.tokenize(text))]
```

The above function returns a dedicated vector of `Token` objects, each of them containing the POS tagged terms. The class used for this is implemented in the next code section.

The `Token` class uses a polarity lexicon, where the positive and negative terms defined in the earlier mentioned text files `positive-words.txt` and `negative-words.txt` are represented.

```
class Token(object):

    polarityLexicon = load_polarity_lexicon(["./resources/positive-words.txt", "./
    resources/negative-words.txt"],
                                            ["positive", "negative"])

    __slots__ = ['text', 'pos', 'positiveScore', 'negativeScore']

    def __init__(self, text, pos=None):
        start = None
        end = None
        self.text = text
        self.pos = pos
        if (self.text in self.polarityLexicon and
                self.polarityLexicon[self.text] == "positive"):
            self.positiveScore = 1
        else:
            self.positiveScore = 0

        if (self.text in self.polarityLexicon and
                self.polarityLexicon[self.text] == "negative"):
            self.negativeScore = 1
        else:
            self.negativeScore = 0
```

On initialization, the current word is checked on its existence in the polarity lexicon. If there is a tendency to positive or negative sentiment, it is likely to be located in it. If the polarity is then approved as positive, the variable `self.positiveScore` is assigned the number one. The same process is applicable to a negative term within the review text. Later on, when the sentiment of the product review is checked, the simple comparison between the two variables `positiveScore` and `negativeScore` is made. Depending on which of the two is higher, the sentiment is determined.

```
def load_polarity_lexicon(filenames, categories):
    # Return a dictionary containing the words from the given files and their
    corresponding category
    assert len(filenames) == len(categories)
    polarityLexicon = {}
    for filename, category in zip(filenames, categories):
        with codecs.open(filename, 'r', encoding='latin-1') as wordsFile:
            polarityLexicon.update({w.strip(): category
                                  for w in wordsFile.readlines()
                                  if w.strip() and not w.strip().startswith(";")})
    return polarityLexicon
```

To load the polarity lexicon, a dictionary is created initially. The given text file is read into the dictionary, word by word. Lines starting with a semicolon are ignored, as these serve as a headline or commentary. The lexicon is then returned and used to grade the terms within the product reviews.

4.2.3 Data Preprocessing

Before training the deep neural network, the review text of each data entry has to be cleaned and preprocessed using different ML techniques. The major component of the project is located within the function `apply_deep_learning()`. At the beginning of the function, the IDs are another time shuffled to change the order of the training and test data entries. Right after that, the CBOW model is created, as illustrated in the code section below.

```
def create_cbow(reviews):
    # Create continuous bag-of-words model
    data = []

    for review in reviews.values():
        for i in sent_tokenize(review.text):
            temp = []

            # Tokenize the sentence into words
            for j in word_tokenize(i):
                temp.append(j.lower())

            data.append(temp)

    # Create CBOW model
    return gensim.models.Word2Vec(data, min_count=1, size=300, window=5)
```

In this function, a data array is created, where all the sentence of all product reviews are tokenized into words and then added. The tokenization is done by the functions `sent_tokenize()` and `word_tokenize()` of the NLTK library. Here, the Gensim framework is used to handle word embeddings by providing the data frame as input to its vectorization function. The context window size can be passed as the parameter `window`. It can also be described as the maximum distance between a target word and words around the target word. The window size of five words exposed as an adequate number for yielding appropriate results. The parameter `size` defines the number of dimensions of the embeddings, where the default value is 100. The minimum count of words is the number to consider when training the model. Words with occurrence less than this count will be ignored. The default for `min_count` is five. The default algorithm used for training is the CBOW method. To train using the skip-gram method, the parameter `sg` can be altered.

The following data preprocessing step is to extract the features of each review. This is done for training and test set individually. The function `feature_extract(review_ids, reviews, bow_dictionary)` is implemented in the following way.

```
def feature_extract(review_ids, reviews, bow_dictionary):
    feature_vec = {}
    for ID in review_ids:
        review = reviews[ID]
        feature_vec[ID] = []

        # Star Rating
        feature_vec[ID].extend(star_rating(review))

        # Sentiment
        feature_vec[ID].extend(sentiment(review))

    return feature_vec
```

First, a Python dictionary is created for the purpose of being filled with review-specific features. The first feature stored in the dictionary is the star rating of each the review currently looked at. The second one is the overall sentiment of this particular product review text.

```
def star_rating(review):
    return [1 if i+1 == int(review.stars) else 0 for i in range(5)]
```

For every product review, a feature array is appended. This array contains the stars given at the product rating but in a one-hot encoded form. A five-star rated product would have the feature array `[0 0 0 0 1]`, while a two-star rated product would be filled in the way `[0 1 0 0 0]`. The second feature is implemented below.

```
def sentiment(review):
    # Return a vector for the sentiment of a given review
    result = [0] * 1
    polarity = len(review.positive_words) - len(review.negative_words)

    if polarity > 0:
        result[0] = 1

    return result
```

Observing the sentiment of a given review is done by inspecting the negative and positive words within the product review text. The previously collected vectors of either negatively or positively tempered words are compared to their length. If more negatively tempered words exist within the review, the overall sentiment is defined as negative. If more positive terms exist, the sentiment is described as positive. The complete feature vector is then returned and can be used for training the DNN.

Stop word removal

Why is removing stop words not always a good idea? Stop words removal is a crucial preprocessing step in many machine learning applications dealing with text. For some applications, stop words are actually needed. A stop word may be considered a word that has high frequency on a corpus, such as articles or terms that do not help finding the true meaning of a sentence. These words can be removed without any negative consequences to the training of the model. These stop words are always different, as it depends on the corpus used.

Reducing the data set size is a way of increasing performance. Training models can be accelerated by removing unwanted tokens. Thus the training time should decrease. As the project's used data collection is rather small, the stop words removal is not needed before training the neural network. Problems like sentiment analysis are much more sensitive to stop words removal than document classification. An example could be the following sentence: “I told you that she was not happy”. After removing the stop words, the result would be ['told', 'happy']. For sentiment analysis purposes, the overall meaning of the resulting sentence is positive. In reality, the sentiment should be negative, but not considered as negative, as the word “not” was removed in the preprocessing step. This is the main reason why stop words removal is not done throughout this research [62].

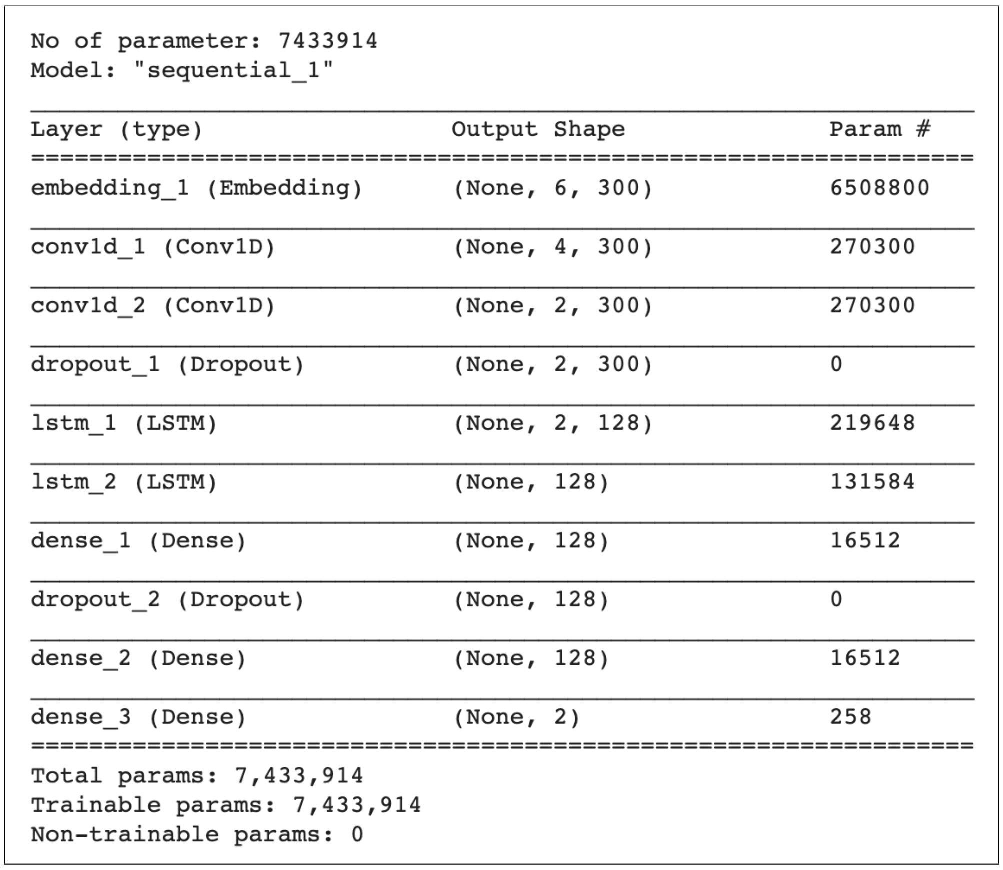
Part-of-speech Tagging

The most basic models are based on BOW in the NLP area. But such models fail to capture the syntactic relations between words. A sentiment analyzer based only on BOW will not be able to capture the difference between “I like you” and “I am like you”. In the first case, “like” is a verb with a positive sentiment, while in the other case “like” is a preposition with a neutral sentiment.

POS tagging is utilized for building parse trees, which are used in extracting relations between words. During the process, a word in a corpus is marked up to a corresponding part of a speech tag, based on its context and definition. An example can be seen in the sentence “Give me your answer”. In this particular case, answer is a noun, but in the sentence “Answer the question”, answer is a verb. To understand the meaning of any sentence, POS tagging is a crucial step.

The different techniques of POS tagging involve lexical base methods on the one hand, where the tag is assigned the most frequently occurring with a word in the training corpus. The rule-based methods assign POS tags based on specific rules, such as endings like “ed” or “ing” must be assigned to a verb. The probabilistic method assigns the tags based on the chance of a particular tag sequence occurring, such as *Conditional Random Fields* (CRFs) and *Hidden Markov Models* (HMMs). The last method is the DL method, which uses RNNs for POS tagging [57].

The tagging of the sentence “They refuse to permit us to obtain the refuse permit” would look like the following vector: `[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]` [4]. The tag “NN” represents a singular noun, “VB” just defines a verb in its base form. “VBP” describes a verb in its singular, present form, while “TO” is just the tag for the term “to”. The tag “PRP” signifies a personal pronoun, such as he, she or I. “DT” is the tag for a determiner. A list of available tags is described in more detail in the appendix, specifically in chapter .



```
No of parameter: 7433914
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 6, 300)            6508800   
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 4, 300)            270300    
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 2, 300)            270300    
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 300)            0         
_________________________________________________________________
lstm_1 (LSTM)                (None, 2, 128)            219648    
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584    
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512     
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512     
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258       
=================================================================
Total params: 7,433,914
Trainable params: 7,433,914
Non-trainable params: 0
```

Figure 4.6: Model summary of the proposed DNN

4.2.4 Deep Learning Model

The implemented DNN is the one initially created by Ghosh and Veale [14]. As mentioned earlier, the neural network consists of two CNNs, two LSTMs, a dense neural network and an activation layer. The summary of the model can be seen in figure 4.6. The neural network was implemented within the method `model_dnn(x_train, y_train, x_test, y_test, cbow=None)`, that holds the entire modelling and all the parameters used for training and testing. The process is described in more detail in the code section below.

```
def model_dnn(x_train, y_train, x_test, y_test, cbow=None):
    embedding_dimension = 300
    hidden_size = 128
    max_len = 6
    vocabulary = len(cbow.wv.vocab)

    # The batch size defines, how many samples the model sees at once - in this case 64 reviews
    batch_size = 64

    # The epoch variable defines, how often the model sees the data set
    epochs = 500
```

The `embedding_dimension` was set to 300, which is one of the most used size numbers for word embeddings. The `hidden_size` was defined as 128. The variable `vocabulary` desribes the size of the CBOW model's vocabulary. At the beginning, a `batch_size` of 64 was used. To be sure, the amount of epochs run was defined as 500. After defining the initial constants, a matrix of embeddings had to be created.

```
# Save the word2vec vectors in a new matrix
embedding_matrix = zeros((len(cbow.wv.vocab), embedding_dimension))
for i, vec in enumerate(cbow.wv.vectors):
    embedding_matrix[i] = vec
```

The Word2vec vector was converted to an `embedding_matrix` containing each element in the vector space. This matrix is fed into the first layer of the neural network, the embedding layer. The DL model is built in the following manner.

```
# Start to build model
model = Sequential()

model.add(Embedding(input_dim=vocabulary, output_dim=embedding_dimension,
input_length=max_len,
                    weights=[embedding_matrix], embeddings_initializer='
glorot_uniform'))

# Reduce frequency variation through convolutional filters and extract
discriminating word sequences as a composite
# feature map for the LSTM layer
model.add(Convolution1D(embedding_dimension, 3, kernel_initializer='
glorot_normal', padding='valid', activation='sigmoid', input_shape=(1, max_len))
)
model.add(Convolution1D(embedding_dimension, 3, kernel_initializer='
glorot_normal', padding='valid', activation='sigmoid', input_shape=(1, max_len -
 2)))

# Dropout layer to avoid overfitting
model.add(Dropout(0.25))

# LSTM layer
model.add(LSTM(hidden_size, kernel_initializer='glorot_normal', activation='
sigmoid', dropout=0.5, return_sequences=True))
model.add(LSTM(hidden_size, kernel_initializer='glorot_normal', activation='
sigmoid', dropout=0.5))

# Output of LSTM layer is passed to a fully connected dense layer
# It produces a higher order feature set based on the LSTM output, which is
easily separable for the desired number
# of classes
model.add(Dense(hidden_size, kernel_initializer='glorot_normal', activation='
sigmoid'))
model.add(Dropout(0.25))

```

```
    model.add(Dense(hidden_size, kernel_initializer='glorot_normal', activation='
     sigmoid'))

    # Final softmax layer
    model.add(Dense(2, activation='softmax'))
```

The model starts with the embedding layer, getting passed the vocabulary length, as well as the output dimension of the length `embedding_dimension`. The embeddings initializer if the function `glorot_uniform`. The uniform algorithm finds a good variance for the distribution from which the initial parameters are drawn. This variance is adapted to the activation function used and derived without explicitly considering the type of distribution. The Word2vec model is actually passed as a vector to the layer function.

Both CNN layers are initialized with the size of the embedding dimension. The kernel initializer used in these cases was the algorithm `glorot_normal`. The same construct is given when adding the two LSTM layers to the neural network. The only difference is the size of the hidden layers, which is 128 in this case. Lastly, a dense layer and a softmax activation layer are added to the network. Softmax functions limit the output of the function into the range 0 to 1. This allows the output to be interpreted directly as a probability. After creating the model structure, the model is compiled using an optimization and a loss function.

```
# Optimization of parameters using ADAM
adam = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0,
amsgrad=False)
sgd = SGD(lr=0.01, momentum=0.9, decay=1e-6, nesterov=True)
adadelta = Adadelta()

# Loss function declarations
scc = 'sparse_categorical_crossentropy'
bcc = 'binary_crossentropy'
cc = 'categorical_crossentropy'
loss = tf.keras.losses.Huber(delta=1.0)

# Model compilation
model.compile(loss=loss, optimizer=adam, metrics=['accuracy'])
print('No of parameter:', model.count_params())
```

At first, the model was trained using the *Adam* optimizer, with a learning rate of 0.0001. The loss function at this point was the function `sparse_categorical_crossentropy`. The metric to look at compilation was just the accuracy value. After compilation, the model is fitted.

```
# Callback functions for saving model
save_best = ModelCheckpoint("./model/" + 'model.json.hdf5', monitor="acc",
verbose=1, save_best_only=True,
                           mode="max")
save_all = ModelCheckpoint("./model/" + 'weights.{epoch:02d}__.hdf5',
save_best_only=False)
early_stopping = EarlyStopping(monitor='val_loss', patience=20, verbose=1)
```

```
    lr_scheduler = LearningRateScheduler(step_decay)

    # Model fitting
    history = model.fit(x_train, y_train, validation_split=0.20, batch_size=
     batch_size, epochs=epochs, shuffle=True,
                        callbacks=[save_best, save_all, early_stopping])
```

The fitting of the model is done using a validation split of 20 percent of the training data. Four different callbacks have been created, where only three of them were used at the beginning. The callback `save_best` offers the ability to save a model into a specific file directory, every time it achieves better accuracy than in the earlier epoch. The same applies for `save_all`. There, the completely fitted model is saved. The `early_stopping` method decides whether to abort a learning process. If there has been no gain in accuracy for 20 epochs, the model fitting is completed.

```
    # Evaluating the current model - it's accuracy
    loss, accuracy = model.evaluate(x_train, y_train, verbose=False)
    print("Training accuracy: {:.4f}".format(accuracy))
    loss, accuracy = model.evaluate(x_test, y_test, verbose=False)
    print("Testing accuracy:  {:.4f}".format(accuracy))
```

As the last step, the model is evaluated according to its performance and accuracy. Therefore, the evaluation is done utilizing training data at first. Afterwards, test data are evaluated. So with this step, the neural network training is completed.

4.2.5 Learning Process

The learning process itself contains several steps for achieving the best results in sarcasm detection within product reviews. The first step is to tune all the hyperparameters of the neural network, which is described in section 4.2.5.

Hyperparameter Tuning

The hyperparameter tuning was first done by hand. Several parameter values were tried out until no performance and accuracy gain was reached. To achieve the best performance for the model, an algorithm was used to find the most fitting hyperparameters. This method is called "Random Search". Random search tries out a random combination of parameters after another. Once a fairly good result is achieved, further searching will take place near the area of these parameters.

```
def random_search(x_train, y_train):
  batch_size = 128
  epochs = 10

  # Wrap Keras classifier to use it with scikit-learn
  keras_model = KerasClassifier(build_fn=create_model, epochs=epochs, batch_size=
    batch_size, verbose=1)

   # Learning algorithm parameters
```

```
lr=[1e-2, 1e-3, 1e-4]
decay=[1e-6,1e-9,0]

# Activation
activation=['relu', 'sigmoid']

# Dropout and regularisation
dropout = [0, 0.1, 0.2, 0.3]
l1 = [0, 0.01, 0.003, 0.001,0.0001]
l2 = [0, 0.01, 0.003, 0.001,0.0001]

loss = ['sparse_categorical_crossentropy', 'tf.keras.losses.Huber()', '
  categorical_crossentropy', 'binary_crossentropy']

param_distributions = dict(act=activation, l1=l1, l2=l2, lr=lr, decay=decay,
  dropout=dropout, loss=loss)

random = RandomizedSearchCV(estimator=keras_model,
                            cv=KFold(3),
                            param_distributions=param_distributions,
                            verbose=20, n_jobs=1,
                            n_iter=10)

random_result = random.fit(x_train, y_train)

print('Best Score: ', random_result.best_score_)
print('Best Params: ', random_result.best_params_)

return random_result
```

The function `random_search(x_train, y_train)` starts with an initialization of the number of epochs and the batch size. For hyperparameter tuning, a relatively small size of ten was chosen as the number of epochs, as well as a moderate size of 128 as the batch size. Through `KerasClassifier()`, the model is wrapped to be used with Scikit-learn. The function `create_model` returns the fully compiled model using various combinations of the parameters. The chosen parameter ranges can be seen in the code section above.

The function `RandomizedSearchCV()` is provided by Scikit-learn and does the job of searching for the most optimal hyperparameter combination to achieve the best results. The estimator passed to the function is the object to be instantiated for each grid point. It is assumed to implement the Scikit-learn estimator interface. The variable `param_distributions` is a dictionary with parameters names as keys and distributions or lists of parameters to try. The variable `n_iter` defines the number of parameter settings that are sampled, while `n_jobs` is the number of jobs to run in parallel. `Verbose` controls the verbosity: the higher, the more messages. The `cv` parameter determines the cross-validation splitting strategy. The possible inputs for `cv` are either None, to use the default 3-fold cross-validation, or any integer, to specify the number of folds in a (Stratified)KFold.

The grid search is another hyperparameter optimization algorithm. The main difference to the random search algorithm is the execution. In the grid search, every possible combination of given parameters is executed while optimizing. The randomized search and the grid search explore exactly the same space of parameters. The result in parameter settings is quite similar, while the runtime for a randomized search is drastically lower [32]. In practice, one would not search over this many different parameters simultaneously using grid search, but pick only the ones deemed most important. This is why the random search was chosen for the project realization.

Parameter Choice

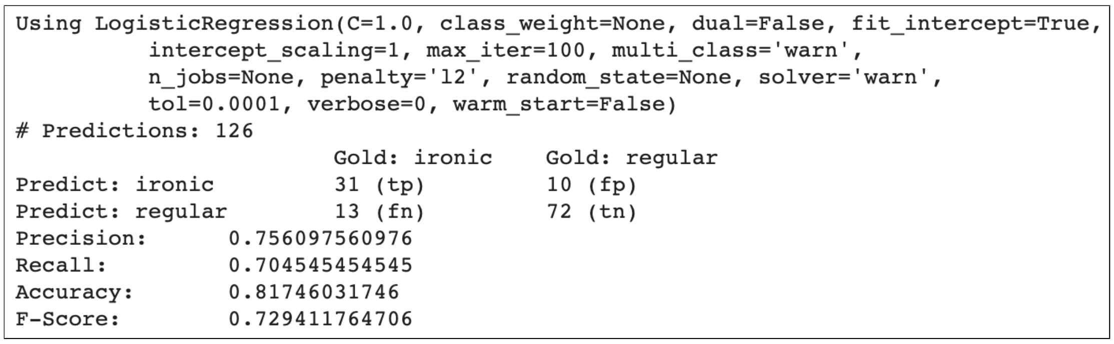
After searching for the most optimal parameter choice using the random search, the following parameters were destined to be taken. A `batch_size` of 256, as well as the number of epochs of 500, have been used for fitting the neural network model. As the batch size was defined rather high, the learning rate had to be rather low, to avoid overfitting of the model.

The currently utilized optimization function is the *Adam* optimization, with an initial learning rate of 0.001. Adam is the adaptive learning rate optimization algorithm specifically designed for training DNNs. Adam is considered a combination of *Root Mean Square* (RMSprop) and *Stochastic Gradient Descent* (SGD) with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the the gradient instead of gradient itself like SGD with momentum. As Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters, the algorithm fits well for the product review sarcasm detection task. Its name is derived from adaptive moment estimation, as it uses estimations of first and second moments of a gradient to adapt the learning rate for each weight of the neural network [22].

The sarcasm detection task is a so-called binary classification problem, which is a problem where one classifies an example as belonging to one of two classes. The problem is framed as predicting the likelihood of an example belonging to class one, the class that one assigns the integer value 1, whereas the other class is assigned the value 0. The output layer of a binary classification problem is a node with a `sigmoid` activation unit. The loss function used for solving this problem is the cross-entropy function, also referred to as Logarithmic loss [16]. In the project's case the `sparse_categorical_crossentropy` is used. The Keras Huber loss was tried out as well, in case it would result in a better accuracy percentage. Indeed, the training and test loss was severely reduced, to a number below ten percent, but the training and test accuracy performed way worse than with using the categorical cross-entropy approach [22].

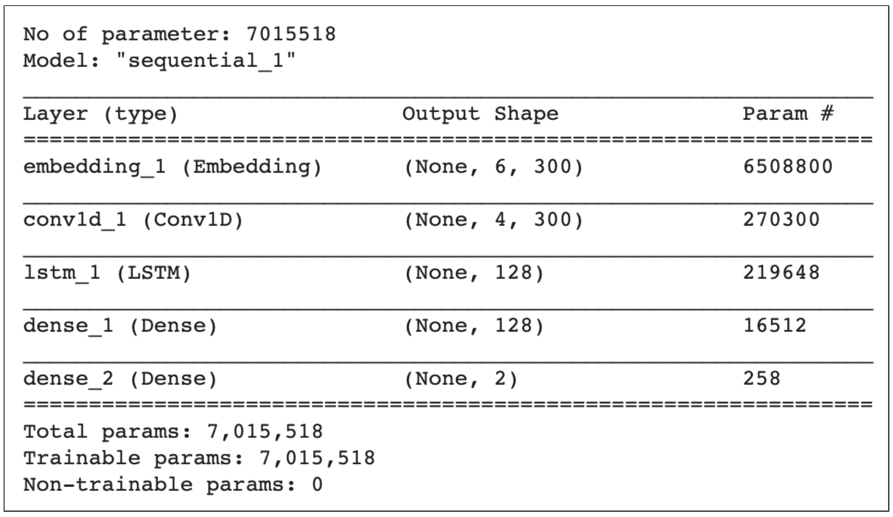
4.3 Results

At the end of the project research, the results of several models have been evaluated and taken into account. The neural network has been trained and tested, and corresponding accuracy value has been returned. For comparability, a more simplistic, standard model was evaluated on the sarcasm review data set as well. This LR model outputs the following results, as can be seen in figure 4.7. The precision of the logistic regression resulted in a percentage of around 75.6. The accuracy is about 81.7 percent. This model shows promising performance when evaluating. Regarding the results of the deep neural network at first, the findings in training and test accuracy were not satisfiable enough.



```
Using LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
# Predictions: 126
                        Gold: ironic    Gold: regular
Predict: ironic         31 (tp)         10 (fp)
Predict: regular        13 (fn)         72 (tn)
Precision:      0.756097560976
Recall:         0.704545454545
Accuracy:       0.81746031746
F-Score:        0.729411764706
```

Figure 4.7: Extract of the LR model results



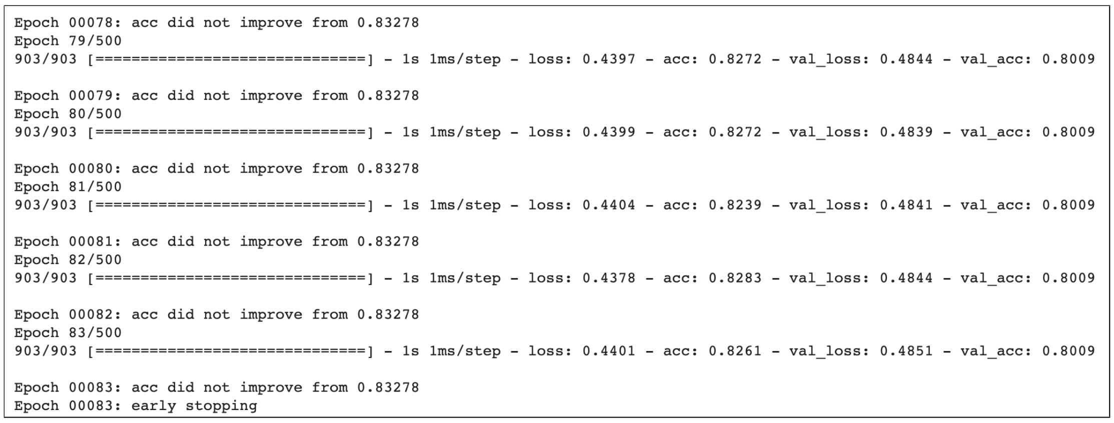
```
No of parameter: 7015518
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 6, 300)            6508800
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 4, 300)            270300
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               219648
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 7,015,518
Trainable params: 7,015,518
Non-trainable params: 0
```

Figure 4.8: Model structure of the cut-down DNN

The training and test accuracy remained stuck in the area of about 65 percent. The simpler, LR model returned higher accuracy values and therefore seemed to perform better than a DNN in the first place.

After cutting down the model to the most needed components, the model structure is illustrated in figure 4.8. As can be seen above, some layers were discarded. The setup of the cut-down model results in a way better training and test accuracy, already outperforming the LR model. As the model training did not improve after about 83.3 percent, early stopping took place. The abortion process is illustrated in figure 4.9. In figure 4.10, the training curve of both the training and test accuracy are displayed. The training accuracy starts rather low at about 66 percent, while increasing drastically to epoch 18.

At this point, the learning settles down to the accuracy of 83 percent. Figure 4.11 thoroughly illustrates the loss of both training and test data over all the executed epochs during the training. Both loss curves show a drastic drop at about ten epochs, while at epoch 20, the loss values stay more or less the same. In comparison to the other model, the implemented neural network does perform well, even considering the rather tight



```
Epoch 00078: acc did not improve from 0.83278
Epoch 79/500
903/903 [==============================] - 1s 1ms/step - loss: 0.4397 - acc: 0.8272 - val_loss: 0.4844 - val_acc: 0.8009

Epoch 00079: acc did not improve from 0.83278
Epoch 80/500
903/903 [==============================] - 1s 1ms/step - loss: 0.4399 - acc: 0.8272 - val_loss: 0.4839 - val_acc: 0.8009

Epoch 00080: acc did not improve from 0.83278
Epoch 81/500
903/903 [==============================] - 1s 1ms/step - loss: 0.4404 - acc: 0.8239 - val_loss: 0.4841 - val_acc: 0.8009

Epoch 00081: acc did not improve from 0.83278
Epoch 82/500
903/903 [==============================] - 1s 1ms/step - loss: 0.4378 - acc: 0.8283 - val_loss: 0.4844 - val_acc: 0.8009

Epoch 00082: acc did not improve from 0.83278
Epoch 83/500
903/903 [==============================] - 1s 1ms/step - loss: 0.4401 - acc: 0.8261 - val_loss: 0.4851 - val_acc: 0.8009

Epoch 00083: acc did not improve from 0.83278
Epoch 00083: early stopping
```

Figure 4.9: Early stopping applying during training of the cut-down DNN

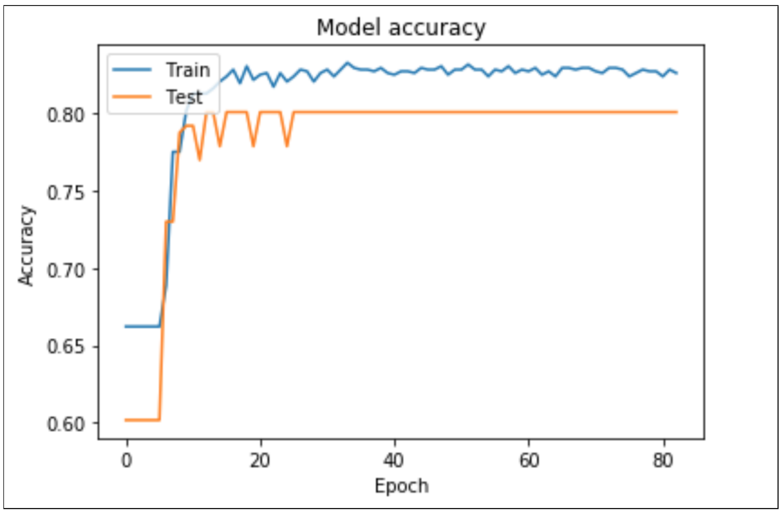


Figure 4.10: Training and test accuracy of the cut-down DNN

data set. All in all, the neural network has great potential to be used with more data. It is also a base model when more features are input. Thus a even better performance may be achieved.

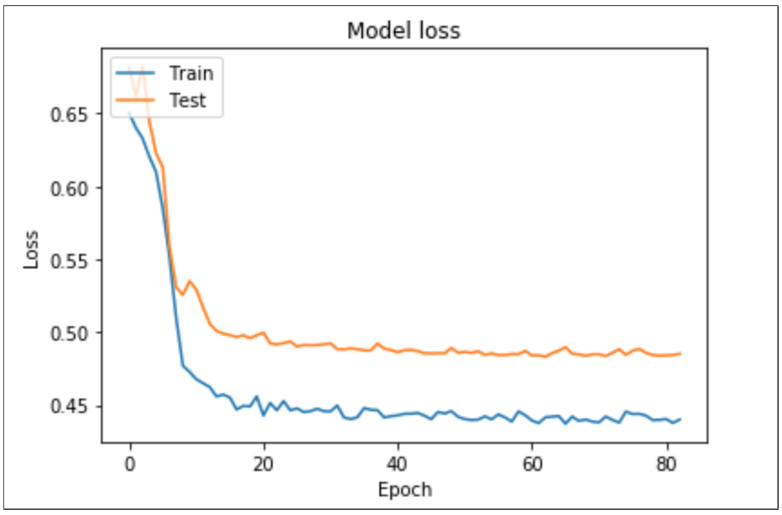


Figure 4.11: Training and test loss of the the cut-down DNN

Chapter 5

Conclusion and Future Work

This thesis work presented the novel idea of deep neural networks composed of CNN and LSTM layers highly probable giving improved results over state-of-the-art discrete models. Sarcasm understanding has become an important topic these days and should be even more investigated to provide a convenient tool for interested parties. Future work should be mindful of already achieved results and therefore, improve performance.

The achieved results are within the range of the expected, given the difficulty of the classification problem. The main goal of the research was to describe a sarcasm detection system, structured as a deep neural network, using a dedicated Amazon product review corpus. Even human experts would disagree about some of the classifications in the corpus. The overall findings during the research comprise the clarification, that sarcasm is majorly used in review, where the particular product is given an overall low star rating.

For future researches, a broader sarcastic review data set shall be used, to build an efficient model acting as a classifier in a wider range of topics. Having such a small amount of data available, a neural network having this many deep layers might not be the right choice for training, as similar results are obtained using standard regression models. Nevertheless, the model layers themselves perform well when working with context-sensitive input. As well as the presence of the Word2vec model made the deep neural network's accuracy achieve higher percentage.

The importance of feature engineering (relative to simply obtaining more data points) can not be over-emphasized. The feature extraction is the part of sarcasm detection where most of the effort should be placed in order for sarcasm recognition to be successful. In the end, the project made it possible to broaden the knowledge in the sentiment analysis field, as well as taking a small step in the right direction.

Appendix A

CD-ROM/DVD Contents

```
A_Deep_Learning_Model_for_Detecting_Sarcasm.pdf
project.zip/
    Amazon_Review_Sarcasm_Detection.ipynb
    data.zip/
        corpus/
            Ironic/
                *.html
                *.txt
            Regular/
                *.html
                *.txt
            five_labels_plus_stars.xlsx
            file_pairing.txt
        resources/
            negative-words.txt
            positive-words.txt
```

Figure A.1: Project content directory tree

Appendix B

List of Acronyms

AI	Artificial Intelligence
API	Application Program Interface
AUC	Area Under the Curve
BOW	Bag-Of-Words
CBOW	Continuous Bag-Of-Words
CNN	Convolutional Neural Network
CNTK	Cognitive Toolkit
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
FPR	False Positive Rate
GPU	Graphics Processing Unit
HDP	Hierarchical Dirichlet Process
HTML	Hypertext Markup Language
ID	Identification
IDE	Integrated Development Environment
IWS	Interjection Word Start
LDA	Latent Dirichlet Allocation
LR	Logistic Regression
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-layer Perceptron
MTurk	Amazon Mechanical Turk
MV	Machine Vision
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
OOP	Object-Oriented Programming
PBLGA	Parsing Based Lexical Generation Algorithm

PD Pattern Detection
PHP PHP: Hypertext Preprocessor
POS Part-Of-Speech
RAM Random Access Memory
ReLU Rectified Linear Unit
RMSprop Root Mean Square
RNN Recurrent Neural Network
RP Random Projections
RPA Robotic Process Automation
SA Sentiment Analysis
SGD Stochastic Gradient Descent
SVM Support Vector Machine
TPR True Positive Rate
TPU Tensor Processing Unit
UI User Interface

Appendix C

Part-of-speech Tags and their Meaning

The following definitions are retrieved from the website *Medium* [56].

CC Coordinating conjunction
CD Cardinal digit
DT Determiner
ET Existential there
FW Foreign word
IN Preposition/subordinating conjunction
JJ Adjective 'big'
JJR Adjective, comparative 'bigger'
JJS Adjective, superlative 'biggest'
LS List marker 1)
MD Modal could, will
NN Noun, singular 'desk'
NNS Noun, plural 'desks'
NNP Proper noun, singular 'Harrison'
NNPS Proper noun, plural 'Americans'
PDT Predeterminer 'all the kids'
POS Possessive ending parent's
PRP Personal pronoun I, he, she
PRP$ Possessive pronoun my, his, hers
RB Adverb very, silently
RBR Adverb, comparative better
RBS Adverb, superlative best
RP Particle give up
TO To go 'to' the store
UH Interjection, errrrrrrrm
VB Verb, base form take
VBD Verb, past tense took

VBG Verb, gerund/present participle taking
VBN Verb, past participle taken
VBP Verb, sing. present, non-3d take
VBZ Verb, 3rd person sing. present takes
WDT Wh-determiner which
WP Wh-pronoun who, what
WP$ Possessive wh-pronoun whose
WRB Wh-abverb where, when

References

Literature

[1] Silvio Amir et al. “Modelling Context with User Embeddings for Sarcasm Detection in Social Media”. In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning.* Berlin, Germany: Association for Computational Linguistics (ACL), Aug. 2016, pp. 167–177 (cit. on p. 16).

[2] David Bamman and Noah A. Smith. “Contextualized Sarcasm Detection on Twitter”. In: *Proceedings of the International AAAI Conference on Weblogs and Social Media.* Oxford, United Kingdom: Association for the Advancement of Artificial Intelligence, 2015, pp. 574–577 (cit. on p. 10).

[3] S. K. Bharti, K. S. Babu, and S. K. Jena. “Parsing-based sarcasm sentiment recognition in Twitter data”. In: *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM).* Paris, France: Institute of Electrical and Electronics Engineers (IEEE), Aug. 2015, pp. 1373–1380 (cit. on p. 11).

[4] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python. Analyzing Text with the Natural Language Toolkit.* Adaptive Computation and Machine Learning series. Sebastopol, California, USA: O’Reilly Media Inc., June 2009, p. 504 (cit. on pp. 31, 46).

[5] Konstantin Buschmeier, Philipp Cimiano, and Roman Klinger. “An Impact Analysis of Features in a Classification Approach to Irony Detection in Product Reviews”. In: *Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis (WASSA).* Baltimore, Maryland, USA: Association for Computational Linguistics (ACL), Jan. 2014, pp. 42–49 (cit. on p. 12).

[6] T. Carneiro et al. “Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications”. *IEEE Access* 6 (2018), pp. 61677–61685 (cit. on pp. 28, 29).

[7] Paula Carvalho et al. “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.* Hong Kong, China: Association for Computing Machinery (ACM), 2009, pp. 53–56 (cit. on p. 12).

[8] Richard Chin. “The Science of Sarcasm? Yeah, Right” (Nov. 2011) (cit. on pp. 26, 27).

[9] Cameron L. Davis et al. “White matter tracts critical for recognition of sarcasm”. *Neurocase* 22.1 (2016), pp. 22–29 (cit. on p. 3).

[10] Tom Fawcett. “ROC Graphs: Notes and Practical Considerations for Data Mining Researchers”. *ReCALL* 31 (Jan. 2004), pp. 1–38 (cit. on p. 11).

[11] Bjarke Felbo et al. “Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing.* Copenhagen, Denmark: Association for Computational Linguistics (ACL), 2017, pp. 1615–1625 (cit. on p. 9).

[12] Elena Filatova. “Irony and Sarcasm: Corpus Generation and Analysis Using Crowdsourcing”. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012).* Istanbul, Turkey: European Languages Resources Association (ELRA), May 2012, pp. 392–398 (cit. on pp. 6, 12, 14, 19, 34, 35).

[13] Laura Geggel. “Why You Get the Joke: Brain’s Sarcasm Center Found”. *Live Science* (Apr. 2015) (cit. on p. 3).

[14] Aniruddha Ghosh and Tony Veale. “Fracking Sarcasm using Neural Network”. In: *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis.* San Diego, California, USA: Association for Computational Linguistics (ACL), June 2016, pp. 161–169 (cit. on pp. 7, 25, 47).

[15] Debanjan Ghosh, Alexander R. Fabbri, and Smaranda Muresan. “Sarcasm Analysis Using Conversation Context”. *Computational Linguistics* 44.4 (Dec. 2018), pp. 755–792 (cit. on p. 16).

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* Adaptive Computation and Machine Learning series. Cambridge, Massachusetts: MIT Press, 2016, p. 178 (cit. on p. 52).

[17] John Haiman. *Talk is Cheap. Sarcasm, Alienation, and the Evolution of Language.* Oxford University Press, 1998, p. 12 (cit. on p. 26).

[18] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780 (cit. on pp. 22–24).

[19] Sahil Jain, Ashish Ranjan, and Dipali Baviskar. “Sarcasm Detection in Amazon Product Reviews”. In: Pune, India: International Journal of Computer Science and Information Technologies, 2018, pp. 108–111 (cit. on pp. 5, 12).

[20] Jihen Karoui et al. “Towards a Contextual Pragmatic Model to Detect Irony in Tweets”. 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).* Beijing, China: Association for Computational Linguistics (ACL), July 2015, pp. 644–650 (cit. on p. 2).

[21] Keras. *Keras: The Python Deep Learning library.* 2019 (cit. on p. 30).

[22] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations (ICLR).* San Diego, California, USA, May 2015, pp. 1–15 (cit. on p. 52).

[23] Roger J. Kreuz and Sam Glucksberg. "How to Be Sarcastic: The Echoic Reminder Theory of Verbal Irony". *Journal of Experimental Psychology: General* 118.4 (1989), pp. 374–386 (cit. on p. 16).

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. Nevada, California, USA: Curran Associates Inc., 2012, pp. 1097–1105 (cit. on p. 22).

[25] Ewa Krzaklewska and Graham Gibbs. "Review: Graham Gibbs (2009). Analysing Qualitative Data". *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 11 (Aug. 2010) (cit. on p. 12).

[26] Christine Liebrecht, Florian Kunneman, and Antal Van den Bosch. "The perfect solution for detecting sarcasm in tweets #not". In: *Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Atlanta, Georgia, USA: Association for Computational Linguistics (ACL), June 2013, pp. 29–37 (cit. on p. 10).

[27] Nick Littlestone. "Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm". *Machine Learning* 2.4 (Apr. 1988), pp. 285–318 (cit. on p. 10).

[28] Diana Maynard and Mark Greenwood. "Who cares about Sarcastic Tweets? Investigating the Impact of Sarcasm on Sentiment Analysis." In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. Reykjavik, Iceland: European Languages Resources Association (ELRA), May 2014, pp. 4238–4243 (cit. on p. 11).

[29] Cara McGoogan. "Oh, great. Robots can now tell when people are being sarcastic" (Aug. 2017) (cit. on p. 9).

[30] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations (ICLR): Workshop Track Proceedings*. Scottsdale, Arizona, USA, May 2013, pp. 1–12 (cit. on pp. 14, 15, 31).

[31] Samaneh Nadali, Masrah Murad, and Nurfadhlina Sharef. *Sarcastic Tweets Detection Based on Sentiment Hashtags Analysis*. Vol. 24. 2. American Scientific Publishers, Feb. 2018, pp. 1362–1365 (cit. on p. 2).

[32] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 52).

[33] F. Perez and B. E. Granger. "IPython: A System for Interactive Scientific Computing". *Computing in Science Engineering* 9.3 (May 2007), pp. 21–29 (cit. on p. 28).

[34] Penny M. Pexman. "How Do We Understand Sarcasm?" *Frontiers for Young Minds* 6 (Nov. 2018), p. 5 (cit. on p. 3).

[35] Bernadette M. Randles et al. "Using the Jupyter Notebook As a Tool for Open Science: An Empirical Study". In: *Proceedings of the 17th ACM/IEEE Joint Conference on Digital Libraries*. Toronto, Ontario, Canada: IEEE Press, 2017, pp. 338–339 (cit. on p. 28).

[36] Radim Řehůřek and Petr Sojka. “Software Framework for Topic Modelling with Large Corpora”. English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: European Languages Resources Association (ELRA), May 2010, pp. 45–50 (cit. on p. 31).

[37] S. G. Shamay-Tsoory, R. Tomer, and J. Aharon-Peretz. “The Neuroanatomical Basis of Understanding Sarcasm and Its Relationship to Social Cognition”. *Neuropsychology* 19.3 (2005), pp. 288–300 (cit. on p. 27).

[38] Stephen Skalicky and Scott Crossley. “Linguistic Features of Sarcasm and Metaphor Production Quality”. In: *Proceedings of the Workshop on Figurative Language Processing*. New Orleans, Louisiana, USA: Association for Computational Linguistics (ACL), June 2018, pp. 7–16 (cit. on p. 2).

[39] Oren Tsur, Dmitry Davidov, and Ari Rappoport. “ICWSM – A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Online Product Reviews”. In: *Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media*. Washington, D.C., Washington, USA: Association for the Advancement of Artificial Intelligence (AAAI), May 2010, pp. 162–169 (cit. on p. 11).

[40] Pascal Vincent et al. “Extracting and Composing Robust Features with Denoising Autoencoders”. In: *Proceedings of the 25th International Conference on Machine Learning*. Helsinki, Finland: Association for Computing Machinery (ACM), 2008, pp. 1096–1103 (cit. on p. 25).

[41] S. G. Wicana, T. Y. İbisoglu, and U. Yavanoglu. “A Review on Sarcasm Detection from Machine-Learning Perspective”. *2017 IEEE 11th International Conference on Semantic Computing (ICSC)* (Jan. 2017), pp. 469–476 (cit. on pp. 16, 26).

[42] Katie Zezima. “The Secret Service wants software that detects social media sarcasm. Yeah, sure it will work.” *The Washington Post* (June 2014) (cit. on p. 4).

Online sources

[43] Daniel Faggella. *What is Machine Learning?* Feb. 2019. URL: https://emerj.com/ai-glossary-terms/what-is-machine-learning/ (visited on 08/18/2019) (cit. on p. 27).

[44] SAS Institute GmbH. *Künstliche Intelligenz. Was es ist und was man darüber wissen sollte*. 2019. URL: https://www.sas.com/de_at/insights/analytics/what-is-artificial-intelligence.html (visited on 03/03/2019) (cit. on p. 17).

[45] SAS Institute GmbH. *Maschinelles Lernen. Was es ist und was man darüber wissen sollte*. 2019. URL: https://www.sas.com/de_at/insights/analytics/machine-learning.html (visited on 03/03/2019) (cit. on pp. 19, 20).

[46] Google. *Frequently Asked Questions*. 2019. URL: https://research.google.com/colaboratory/faq.html (visited on 07/21/2019) (cit. on p. 29).

[47] Shashank Gupta. *Sentiment Analysis: Concept, Analysis and Applications*. Jan. 2018. URL: https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17 (visited on 04/16/2019) (cit. on p. 20).

[48] Dhruvil Karani. *Introduction to Word Embedding and Word2Vec*. Sept. 2018. URL: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa (visited on 12/28/2018) (cit. on p. 7).

[49] Ajay M. Koyimuttal. *What are some of the best sarcastic tweets?* May 2018. URL: https://www.quora.com/What-are-some-of-the-best-sarcastic-tweets (visited on 03/12/2019) (cit. on p. 5).

[50] Merriam-Webster. *Definition of artificial intelligence*. 2019. URL: https://www.merriam-webster.com/dictionary/artificial%5C%20intelligence (visited on 04/26/2019) (cit. on p. 17).

[51] Merriam-Webster. *Definition of sarcasm*. 2019. URL: https://www.merriam-webster.com/dictionary/sarcasm (visited on 03/18/2019) (cit. on p. 2).

[52] MonkeyLearn. *Sentiment Analysis*. 2019. URL: https://monkeylearn.com/sentiment-analysis/ (visited on 01/03/2019) (cit. on pp. 9, 21).

[53] Dr. Rutu Mulkar. *Natural Language Processing vs. Machine Learning vs. Deep Learning*. Aug. 2016. URL: https://rutumulkar.com/blog/2016/NLP-ML (visited on 03/25/2019) (cit. on p. 1).

[54] Michael Nguyen. *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. Sept. 2018. (Visited on) (cit. on pp. 23, 24).

[55] Vibhor Nigam. *Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning*. Sept. 2018. URL: https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90 (visited on 04/14/2019) (cit. on pp. 21–23).

[56] Gianpaul Rachiele. *Tokenization and Parts of Speech (POS) Tagging in Python's NLTK library*. Feb. 2018. URL: https://medium.com/@gianpaul.r/tokenization-and-parts-of-speech-pos-tagging-in-pythons-nltk-library-2d30f70af13b (visited on 08/21/2019) (cit. on p. 60).

[57] Aiswarya Ramachandran. *NLP Guide: Identifying Part of Speech Tags using Conditional Random Fields*. Oct. 2018. URL: https://medium.com/analytics-vidhya/pos-tagging-using-conditional-random-fields-92077e5eaa31 (visited on 08/12/2019) (cit. on p. 46).

[58] Margaret Rouse. *Definition: Künstliche Intelligenz (KI)*. URL: https://whatis.techtarget.com/de/definition/Kuenstliche-Intelligenz-KI (visited on 04/27/2019) (cit. on pp. 18, 20).

[59] George Seif. *An Introduction to Scikit Learn: The Gold Standard of Python Machine Learning*. Sept. 2018. URL: https://towardsdatascience.com/an-introduction-to-scikit-learn-the-gold-standard-of-python-machine-learning-e2b9238a98ab (visited on 08/04/2019) (cit. on p. 31).

[60] TensorFlow. *Why TensorFlow*. URL: https://www.tensorflow.org/ (visited on 08/02/2019) (cit. on p. 30).

[61] TutorialsPoint. *Python 3 Tutorial*. URL: https://www.tutorialspoint.com/python3/index.htm# (visited on 07/26/2019) (cit. on p. 29).

[62] Wilame Lima Vallantin. *Why is removing stop words not always a good idea*. Jan. 2019. URL: https://medium.com/@wilamelima/why-is-removing-stop-words-not-always-a-good-idea-c8d35bd77214 (visited on 08/21/2019) (cit. on p. 46).

[63] Winfried W. Wilcke. *Uranium Ore*. URL: https://www.amazon.com/gp/customer-reviews/R2H5Y58JM1IC4V/ref=cm_cr_arp_d_rvw_ttl?ie=UTF8&ASIN=B000796XXM (visited on 03/26/2019) (cit. on p. 4).

[64] Adil Yitiz. *Uranium Ore*. URL: https://www.amazon.com/gp/customer-reviews/R1NY23RMQJDZYP/ref=cm_cr_arp_d_rvw_ttl?ie=UTF8&ASIN=B000796XXM (visited on 03/26/2019) (cit. on p. 4).