CHALMERS
UNIVERSITY OF TECHNOLOGY

A deep learning approach for identifying sarcasm in text

Bachelor's thesis in Computer Science and Engineering

OSCAR BARK
ANDREAS GRIGORIADIS
JAN PETTERSSON
VICTOR RISNE
ADELE SIITOVA
HENRY YANG

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Bachelor of Science Thesis

A deep learning approach for identifying sarcasm in text

Testing and analysis of recurrent and convolutional neural networks

OSCAR BARK
ANDREAS GRIGORIADIS
JAN PETTERSON
VICTOR RISNE
ADELE SIITOVA
HENRY YANG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
University of Gothenburg
Chalmers University of Technology
Gothenburg, Sweden 2017

A deep learning approach for identifying sarcasm in text
Testing and analysis of recurrent and convolutional neural
OSCAR BARK
ANDREAS GRIGORIADIS
JAN PETTERSSON
VICTOR RISNE
ADELE SIITOVA
HENRY YANG

Supervisor: Mikael Kågebäck
Examiner: Richard Johansson

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg 2017

A deep learning approach for identifying sarcasm in text

OSCAR BARK
ANDREAS GRIGORIADIS
JAN PETTERSSON
VICTOR RISNE
ADELE SIITOVA
HENRY YANG
Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg

Bachelor of Science Thesis

Abstract

The aim of this work is to evaluate the performance of deep learning, specifically models of Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN), on the problem of detecting sarcasm in tweets. This is done partly by comparing our results to current state-of-the art performance, and partly by making a qualitative analysis of network functionality. In addition to this, we also conduct a survey to examine the human ability to detect sarcasm in tweets for result comparison.

We examine three models: Two RNNs, one with Long Short Term Memory (LSTM) cells and one with Gated Recurrent Unit (GRU) cells, and also a CNN. Sarcasm detection is done by binary classification on the same datasets used by related works, and our performance is then compared to that of those works'.

The main questions we aim to answer by analyzing the network functionality are what features a□ect the outcome, and how. By comparing our classifications with those of a basic bag-of-words model, scrambling the word content in tweets and looking at repeatedly misclassified tweets we are able to get a deeper understanding of the networks' decisions.

Experimental results suggest that the networks' predictions mainly are based on word occurrence in the tweets. The best performance reach an F1-score of 0.842 when using the RNN with LSTM-cells. This network performed better overall among our models, indicating it might be the best option for this particular task.

When conducting the survey, the model performed with an F1-score of 0.775 whereas humans reached an average score of 0.701. The model also performed better than a basic bag-of-words model, indicating that deep neural networks might be a feasible approach in tackling the problem of sarcasm detection in text.

Keywords: sarcasm detection, deep learning, RNN, CNN.

Sammanfattning

Målet med projektet har varit att undersöka hur väl deep learning, i synnerhet Recurrent Neural Networks (RNN) och Convolutional Neural Networks (CNN), fungerar för sarkasmdetektion i tweets. Detta görs dels genom att jämföra våra resultat mot nuvarande state of the art och dels genom att utföra kvalitativa analyser av nätverkens funktionalitet utifrån olika experiment. Utöver detta genomför vi även en undersökning för att ta reda på människors förmåga att detektera sarkasm i jämförelse med vårt bäst presterande nätverk.

Vi undersöker tre modeller: Två RNN, ett med Long Short-Term Memory (LSTM) celler samt ett med Gated Recurrent Unit (GRU) celler, och ett CNN. Binär klassifikation används för att detektera sarkasm på samma datasets som använts av liknande arbeten, för att sedan jämföra våra resultat med deras.

De huvudsakliga frågorna vi ämnar att besvara genom att analysera nätverkens funktionalitet är vilka egenskaper som påverkar resultatet och hur. Detta gör vi genom att jämföra våra klassifikationer mot dem hos en grundläggande bag-of-words modell, blanda ordinnehållet i tweets och undersöka återkommande felklassificerade tweets.

Våra slutsatser indikerar på att modellens förutsägelser baseras huvudsakligen på förekomsten av ord hos tweets. Den bästa prestandan når ett F1-score på 0.842 vid användning av modellen RNN med LSTM-celler. Denna modell presterade bäst bland alla de tre modellerna som undersöktes.

När undersökningen för mänsklig sarkasmdetektion genomfördes, presterade vår modell med ett F1-score på 0.775, medan människor nådde medelpoängen 0.701. Modellen presterade även bättre än en grundläggande bag-of-words modell, vilket ger en indikation på att deep learning modeller kan vara en möjlig ingångspunkt i problemet att detektera sarkasm i text.

Nyckelord: Sarkasmdetektion, deep learning, RNN, CNN.

Acknowledgements

Mainly, we would like to thank our supervisor Mikael Kågebäck for guidance throughout the course of the project. We would also like to thank Tomáš Hercig, former Tomáš Ptáček, creator of two of the datasets used in this project and also taking his time to answer our concerns over email. Finally we would like to show appreciation to the people of the department for technical language for helping us with our report.

Glossary

ANN: Artificial Neural Networks
Bag-of-words model: A rule based classification method where the occurrence, or frequency, of words are the features.
Base case: In this thesis, we refer to the best performing network on a certain dataset as the *base case*.
Class: A subset of the dataset, for example: sarcastic and not sarcastic tweets.
CNN: Convolutional Neural Network archetype, a neural network often used in image classification.
Corpus: A synonym for a collection of text, in our case the dataset.
Dataset: A set of datapoints which together make up the dataset.
Dropout: A method to prevent overfitting. Weights are randomly set to 0 according to the dropout rate.
F1-score: A measure of a tests accuracy suitable for unbalanced datasets.
Feature: Measurable property of data.
Generalizability: Ability to perform well on unknown data.
GRU: Gated Recurrent Unit, a special type of activation in a RNN.
Hyperparameters: Adjustable parameters which affect the training or the model.
Key feature: A feature which is outweighs other features in a decision.
Label: See class.
LSTM: Long Short-Term Memory, a special type of activation in a RNN.
Majority class: The class which makes up the majority of a dataset.
Maximum Entropy: A classification method, also known as multinomial logistic regression.
Overfitting: Training a model to overly specialize on a specific set of data, losing generalizability.
Regularization: A method to prevent overfitting. A regularization term proportional to a norm of the weights are added to prevent them from growing too large.
RNN: Recurrent Neural Network, a neural network often used for processing sequential information, e.g. text.
Saddle point: A stationary point where a function is a minima with respect to some variables and a maxima with respect to others.
Scoring method: The metric by which the performance of a classifier is measured.
Supervised learning: Optimizing a model to fit a set of given data.
SVM: Support Vector Machine, a supervised learning model used primary in binary classification analysis.
Tensorflow: A package of API's for implementing machine learning methods.
Unsupervised learning: Inferring a function to reveal hidden structure in unlabeled data.
Vanilla: Default, unextended version.

Contents

List of figures xii

List of tables xiv

1 Introduction 1
1.1 Purpose . . . 2
1.2 Scope . . . 2
1.3 Related works . . . 3
1.4 Contributions . . . 4
1.5 Roadmap . . . 4

2 Artificial Neural Networks 5
2.1 General Artificial Neural Networks . . . 6
2.2 Convolutional Neural Networks . . . 9
2.2.1 Convolution . . . 9
2.2.2 Max pooling . . . 10
2.3 Recurrent Neural Networks . . . 11
2.3.1 Long Short-Term Memory . . . 13
2.3.2 Gated Recurrent Unit . . . 14
2.4 Training . . . 15
2.4.1 Cost function . . . 16
2.4.2 Gradient Descent . . . 16
2.4.2.1 Adaptive moment estimation . . . 17
2.4.3 Backpropagation . . . 19
2.5 Overfitting . . . 20
2.6 Scoring methods . . . 22
2.6.1 Accuracy . . . 23
2.6.2 F1-score . . . 23
2.7 Word embeddings . . . 24

3 Method 25
3.1 Datasets . . . 25
3.1.1 Data preprocessing . . . 26
3.2 Description of used models . . . 27
3.3 Hyperparameter optimization . . . 28
3.4 Training and testing models . . . 28
3.5 Analysis of network functionality . . . 28

3.5.1 Bag-of-words model 29
3.5.2 Word scrambling 29
3.5.3 Trouble makers 30
3.5.4 Impact of word embeddings 30
3.6 Human ability to detect sarcasm 30

4 Results and discussion 32
4.1 Training and testing the models 33
4.2 Analysis of network functionality 36
4.2.1 Bag-of-words model 36
4.2.2 Word scrambling 37
4.2.3 Trouble makers 37
4.2.4 Impact of word embeddings 40
4.3 Human ability to detect sarcasm 41
4.4 Quality of the datasets 42
4.5 Obstacles 44
4.5.1 Previous experience 44
4.5.2 Finding quality data 44
4.6 Impact on society 44

5 Conclusion 45

References 46

A Appendix I

List of gures

2.1 The picture describes the activation of a single neuron. Neighbouring neurons provide inputs of di erent strength, represented by the size of the neurons and the thickness of their connections. These inputs are summed up and determines the activation of the neuron. 6
2.2 A visual representation of a general neural network. A multidimensional input is put into the rst layer of neurons, called the input layer, where it is represented as a vecto$a^{(1)}$. The signal at the next layer, $a^{(2)}$, is formed by two operations. First a weighted linear combination of individual inputs from the rst layer is formed, by matrix multiplication $W^{(1)}a^{(1)} = z^{(1)}$. The signal is then put through an activation function, $f(z^{(1)}) = a^{(2)}$. This procedure may repeat itself for several layers until it produces an output vector whose elements are in the interval [0,1]. 8
2.3 Figure based on gure 9.1 from Goodfellow et al. (2016). The gure shows how the convolution operation works. 10
2.4 Image depicting max pooling. The value of the pooled cells are the maximum of 2x2 cells in the initial layer. 11
2.5 A simple model of a recurrent hidden layeh where the black box indicates a delay for each and every time step. 11
2.6 A RNN unfolding over time where each word is a input at di erent time steps. The colored geometrical shapes mimics the conservation of contextual information. 12
2.7 Adopted from Graves (2008), the gure visualizes with shading that early context have less impact in later time step. Thus the rst input have no impact on the output since new input overwrite the activation of hidden unit. 12
2.8 Figure of a LSTM, taken with permission from Chung et al. (2014). . 14
2.9 Figure of a GRU, taken with permission from Chung et al. (2014). . . 15
2.10 Figure demonstrating optimization paths taken by the vanilla gradient descent and Adam algorithm, both starting from the same initial point. The blue line represents GD, while the red line represents Adam. Note that Adam in this gure converges to the global minimum whereas vanilla GD converges to a local due to the moment property of Adam. 18

2.11 The picture shows two di erent classi ers, black and purple curves, for predicting dots and crosses based on position. The black curve depicts the overall trend among the datapoints in the training set, which is preferable for classifying future data. The purple curve represents an overly complex model which is perfectly adjusted to the training data but is unfavorable for classifying future datapoints. In the latter case the model is said to be over tted. 21
2.12 The picture describes the e ect of over tting. After an initial decrease of both the Test set's and Cross validation set's Cost function, the latter tends to increase due to loss of generality in the classi er. The early stopping method can be used to prevent this, where the training ends at the minima of the Cost function. 22

3.1 The implemented LSTM/GRU network using 256 LSTM/GRU cells, 20,000 vocabulary size, word embeddings of size 200, and 75 time steps. 27

4.1 This gure displays all the F1-scores, and the generalizability across datasets when trained on D3 and tested on D1 ranging from Ptá£ek et al. (2014), Joshi et al. (2015), Poria et al. (2016) to our results. . . 35
4.2 Histogram of tweets ranging from always correctly classi ed (0) to tweets that were always wrongfully classi ed (1). 38
4.3 Zoomed y-axis of the previous gure 4.2's histogram of tweets ranging from always correctly classi ed (0) to tweets that were always incorrectly classi ed (1). 39
4.4 Histogram of human performance, showing the number of participants with a certain F1-score. 42

A.1 Every F1-score that we gathered during to our result and discuss section. II

List of tables

3.1 Table describing the handling of tags 26
3.2 Overview of the used models. 27
3.3 Hyperparameters used while conducting the hyperparameter search ranging from the lowest to the highest value. 28

4.1 Chosen hyperparameters, used for all conducted training experiments in this work. 32
4.2 Result table for the RNN with LSTM-cells where the metrics are represented in F1-score. 33
4.3 Result table for the RNN with GRU cells where the metrics are represented in F1-score. 33
4.4 Result table for the CNN where the metrics are represented in F1-score. 34
4.5 Comparison of F1-scores with related works on di erent datasets. . . 35
4.6 Word frequencies of occurrence in the interval [-1,1] for tags in D1, where 1 and -1 represents sarcastic and non sarcastic respectively. . . 36
4.7 Result table for the bag-of-words model where the metrics are represented in F1-score. 36
4.8 Result table for the RNN with LSTM-cells trained on the original datasets and evaluated on both the original and word scrambled test partitions. 37
4.9 Result table for the RNN with GRU-cells trained on the original datasets and evaluated on both the original and word scrambled test partitions. 37
4.10 Result table for the CNN trained on the original dataset and evaluated on both the original and word scrambled test partitions. 37
4.11 10 randomly selected troublemaker samples which were never classi- ed correctly by the 30 networks trained on D1 with di erent initial weights. 40
4.12 Results for randomly initialized word embeddings versus pretrained GLoVe or Word2Vec model compared with Poria et al. (2016). 41
4.13 Average F1-score of 110 survey participants vs. F1-score on LSTM network classi cations on the same dataset. Standard deviation of the neural network performance is based on 30 di erent runs. 42
4.14 Example of awed sample tweets from our datasets 43

1

Introduction

At least as far back as written history extends, humans have used language to express emotions and share information (World, 2017). Language is however more than its constituent parts of words and sentences, often lled with nuances and insinuations that convey meaning that on the surface can seem contradicting or otherwise hard to discern. To e ectively express ourselves we therefore often use additional means of communication, such as our tone of voice or facial expression. A dramatic example of contradicting sentiment can be found in the use of sarcasm.

The Cambridge (2017) dictionary de nes sarcasm as remarks that mean the opposite of what they say, made to criticize someone or something in a way that is amusing to others but annoying to the person criticized. When used in text, the intended sarcastic meaning is easily lost, due to the lack of some of the previously mentioned queues, otherwise present in speech. This presents problems not only for everyday written communication, but also for people studying language, as in the eld of Natural Language Processing (NLP) (Ptá£ek et al., 2014).

Chowdhury (2005) describes NLP as an area of research dealing with how computers interact with, and understand, natural human language. This ability can be useful for performing tasks in a wide ranging eld of technologies. Di erent research areas include computer- and information Sciences, linguistics, mathematics, robotics and arti cial intelligence. There are many practical applications of NLP, including Machine Translation, Multilingual and Cross-language information retrieval (CLIR). Microsoft Cortana, Apple Siri and Google Assistant are examples of existing software applications and services that make use of NLP.

In this work we are interested in detecting sarcasm speci cally in short text. But why detect sarcasm in text? In the eld of NLP many tasks require understanding the sentiment of a given text (Ptá£ek et al., 2014). According to Poria et al. (2016), sarcasm is key for sentiment analysis and has the ability to completely overturn the polarity of the underlying sentiment.

Other than pure academic interest, there exists a commercial demand. For instance, in 2014 a work order from The Secret Service stated that they were looking for analytic software where one of the requirements were the following: ability to detect sarcasm and false positives (Service, 2014).

The majority of work prior to the current state of the art has relied on language-specific lexical resources (Ptáček et al., 2014). In this work however, we have used deep learning, an approach that has shown great promise in recent years (Poria et al., 2016). Deep learning is a burgeoning field of Machine Learning, using deep neural networks. Neural networks are used for their ability to learn generalized representations of data, meaning that they do not merely memorize information, but learn the underlying trend or concept represented. A deep network, having many layers, gives the ability to learn generalized representations on many different levels of abstraction (Deng and Yu, 2014). For this report, we have implemented and evaluated three deep neural network models for the task of sarcasm detection: Two Recurrent Neural Networks with different activation cells and one Convolutional Neural Network

1.1 Purpose

The purpose of this thesis was to evaluate the performance of deep learning, specifically by implementing models for recurrent neural networks (RNN) and convolutional neural networks (CNN), on the problem of detecting sarcasm in text. To do this, we have performed three main tasks:

- A comparison of the results was made to those of current state of the art methods as reviewed in recent scientific papers. By accessing the same datasets used or developed by other authors, the neural network performance was evaluated by the same scoring methods.
- A comparison of network performance was also made to the human ability to detect sarcasm in tweets, which was examined by conducting an online survey.
- A qualitative analysis of the networks' functionality, i.e. which features affect the classification and how, and performance was made. The latter includes a comparison to a bag-of-words model, which is commonly used due to its simplicity and often high accuracy (Le and Mikolov, 2014).

1.2 Scope

The scope of the thesis is limited to be comparable to the current state of the art in the field of sentiment analysis, as it pertains to sarcasm detection. English is the primary focus of study in the field, and as sentiment such as sarcasm is dependent on language and culture (Joshi et al., 2016a), the datasets will be constrained to the English language. Sarcasm is also very much dependent on context. Therefore this thesis will limit itself to data in contained short text form. Twitter is the most commonly used data source in related works on short text, and will be used in this work accordingly.

The datasets used by Poria et al. (2016), see section 1.3, meet the criteria of our scope. Thus these three datasets were used for the report, and our results were compared to those of Poria et al. (2016).

1.3 Related works

Sarcasm detection in sentiment analysis is a rather new subject that only recently has gained popularity according to Poria et al. (2016). Throughout the years several works have been accomplished using di erent techniques. Joshi et al. (2016b) presented a table with numerous past works summarizing their datasets, approaches, annotations, features and context. The following works of those we have found, are seen as state-of-the-art.

Ptá£ek et al. (2014) tackled the problem using Maximum Entropy and SVM as classi ers in their paper Sarcasm Detection on Czech and English Twitter. They researched on two languages, English and Czech, gathering their data from Twitter. The English datasets, consisting of one balanced and one imbalanced, were obtained through collecting all tweets with the hashtag #sarcasm to indicate sarcastic tweets. Each English dataset contains 100,000 tweets, where the rst dataset is balanced (50,000 sarcastic tweets and 50,000 regular tweets) and the second dataset is imbalanced (25,000 sarcastic tweets and 75,000 regular tweets). On the balanced dataset they acquired an F1-score of 0.946 and on the imbalanced they acquired an F1-score of 0.924.

A di erent approach was used by Poria et al. (2016) in their paper A Deeper Look into Sarcastic Tweets Using Deep Convolutional Neural Networks. As the title suggests, they used neural networks for the task, speci cally deep CNNs. They extracted key features from the CNN and later used the features as input in a SVM for the nal classi cation. For their experiment, they used three di erent datasets containing sarcastic and non-sarcastic tweets. Two of the datasets were obtained from Ptá£ek et al. (2014) and the third was obtained from The Sarcasm Detector [1]. They acquired an F1-score of 0.977 using a baseline CNN and three pretrained features: emotion, personality and sentiment. Their baseline method alone without the pretrained features acquired an F1-score of 0.95 on the balanced dataset from Ptá£ek et al. (2014) when using CNN.

The neural network approach was also used by Ghosh and Veale (2016), in their paper Fracking Sarcasm using Neural Network, who used a neural network model composed of a CNN, followed by a LSTM and ending with a DNN obtaining an F1-score of 0.92. They compared this neural model with a recursive SVM. They also evaluated the performances of the networks individually, CNN+CNN (F1-score: 0.872) and LSTM+LSTM (F1-score: 0.879), where e.g LSTM+LSTM means that the networks are on top of each other. Apart from testing their models on a dataset they built themselves containing 39,000 tweets they also evaluated their system with two available datasets from Rilo et al. (2013) and Davidov et al. (2010).

Two of the three mentioned works above used the neural network approach similarly to us. The exception is Ptá£ek et al. (2014) who instead used maximum entropy and SVM as classi ers.

[1] http://thesarcamsdetector.com

The difference between our tested models and the ones used in Ghosh and Veale (2016) is that our models are not as complex as theirs. As mentioned above Ghosh and Veale (2016) used a model of several networks on top of each other, whereas we test CNN and RNN individually with only one hidden layer. Though, Ghosh and Veale (2016) also evaluated the performance for each network individually as stated above. Another difference is that unlike Ghosh and Veale (2016) who only uses LSTM in their RNN, we make use of both LSTM and GRU in our RNNs.

1.4 Contributions

Several models have been proposed within the field of sarcasm detection. One of the proposed models in this work, the RNN with GRU-cells, has never been used before for this task. Our use of this model, especially in comparison to the RNN model with LSTM-cells, might thus contribute to a general evaluation of its performance.

In addition to this, a part of our work has been centered around doing a qualitative analysis of the network functionality. It's a common perception that, while deep learning works incredibly well, no one really knows how or why (Knight, 2017). None of the papers reviewed in Joshi et al. (2016b) did any kind of analysis as to what the function of their network was. In an effort to contribute to a better understanding of neural networks within this field, a few experiments have been conducted as described in section 3.5.

As a final contribution, we have conducted an experiment to examine human ability to detect sarcasm in tweets. While this has been done before by González-Ibáñez et al. (2011), they only used three people in the test group. Our thesis has done the experiment on a larger scale, with a total of 219 people participating, which might yield a better indication of general human ability.

1.5 Roadmap

The first chapter gives an introduction of the project. Chapter two covers the theoretical background for this project, describing the concept of an Artificial Neural Network and the notion of training a network. Chapter three explains the used methods. The finishing chapters cover the results, discussion and conclusion of our project.

2

Arti cial Neural Networks

In this section the concept of Arti cial Neural Networks (ANN) and model archetypes used for deep learning are thoroughly described. Initially a simple and intuitive description is given while subsequent subsections deals with the mathematics, details and submethods of this subject. Additionally, how a network model is trained with respect to a set of data is covered.

ANN refers to a general eld of computational methods which is inspired by the brain's function of processing information, using a cluster of neurons (McCulloch and Pitts, 1943; Rosenblatt, 1958; Rumelhart et al., 1986). Given a certain multidimensional input, ANN acts upon this input and produces some kind of output. Depending on the application, this output can range from a continuous probability distribution for some quantity, like the weight of an elephant, to a categorization of a certain property, for example if a patient is sick or not (Bridle, 1990). The actual model consists of layers, rather than clusters, of arti cial neurons connected similarly to the brain's axons.

The activation, or output, of a certain neuron is dependent on a linear combination of neighboring neurons' activations, as well as the strength of the connections between them, see gure 2.1. These connections determines the qualities of the network. Adapting them to a certain trainig stimulus is essentially what gives the model the ability to learn. In general, the structure of the network is in the form of multiple neuron layers, where outputs of one layer becomes the input to the next. The input to the network goes to a rst layer, called the input layer , where each neuron in turn is connected to each of the neurons in the next layer, through axons (Kriesel, 2007).

Figure 2.1: The picture describes the activation of a single neuron. Neighbouring neurons provide inputs of di erent strength, represented by the size of the neurons and the thickness of their connections. These inputs are summed up and determines the activation of the neuron.

2.1 General Arti cial Neural Networks

The previous section's intuitive concepts forms the basis for the mathematical model in ANN. In this section, the model is described in a general sense while the following subsections deal with some of the speci c archetypes commonly used in practice.

The signal at each of the layers is represented by a vector $a^{(i)}$ with the same dimension as the number of neurons in the layer, and indexed by the corresponding layer's number (i). The k:th neuron in the i:th layer is denoted $a_k^{(i)}$. The activation of a single neuron is determined by two separate operations. First, a weighted linear combination $z^{(i+1)}$ is formed from the n neurons in the previous layer, with the connecting axons representing these weights $w_{jk}^{(i)}$. The subscripts denotes a connection between the j:th neuron in the rst layer and the k:th one in the second layer. A bias term $a_0^{(i)}$ is also added to the expression, which represents a constant i.e. not dependent on any neutron.

$$z_k^{(i+1)} = w_{0k}^{(i)} a_0^{(i)} + w_{1k}^{(i)} a_1^{(i)} + w_{2k}^{(i)} a_2^{(i)} + \cdots + w_{nk}^{(i)} a_n^{(i)} \tag{2.1}$$

Then an activation function f operates on each element of $z^{(1+i)}$, which produces the output from the neuron. It is common to choose a logistic activation function, in which case $f(z_j)$ produces an output in the range $(0; 1)$. The activation is thus determined by equation (2.2).

$$a_k^{(i+1)} = f(z_k^{(i+1)}) = f(a_0^{(i)} + w_{1k}^{(i)} a_1^{(i)} + w_{2k}^{(i)} a_2^{(i)} + \cdots) \tag{2.2}$$

The function of the whole axon cluster between two layers is to take a vector of n elements $a^{(i)} \in R^n$ as input and produce a vector of m elements $z^{(i)} \in R^m$ as output.

This is done with multiplication between $a^{(i)}$ and a weight matrix $W^{(i)} \in \mathbb{R}^{(m \times n)}$ according to equation (2.3)

$$W^{(i)} a^{(i)} = z^{(i+1)} \tag{2.3}$$

where the weight matrix has the form:

$$W^{(i)} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1} & \cdots & \cdots & w_{mn} \end{pmatrix}$$

containing the weights for all individual axons. The purpose of the activation function mentioned above is to control the degree of activation for a neuron. This function can technically have any form but generally it is continuous and has an s shape with arguments on the reel line. Common choices for this is the sigmoid function and tanh function. More recently, the Rectified linear unit function (ReLU) is also becoming a choice of use in the hidden layers (Glo Dahl et al. (2013)). By indexing the layers from one and upwards equation 2.4 is yielded for the equivalent vectors and matrices.

$$a^{(2)} = f(z^{(2)}) = f(W^{(1)} a^{(1)}) \tag{2.4}$$

Figure 2.2: A visual representation of a general neural network. A multidimensional input is put into the first layer of neurons, called the input layer, where it is represented as a vector $a^{(1)}$. The signal at the next layer, $a^{(2)}$, is formed by two operations. First a weighted linear combination of individual inputs from the first layer is formed, by matrix multiplication $W^{(1)}a^{(1)} = z^{(1)}$. The signal is then put through an activation function, $f(z^{(1)}) = a^{(2)}$. This procedure may repeat itself for several layers until it produces an output vector whose elements are in the interval [0,1].

In brief, a general ANN takes an input in the form of a vector a which is multiplied with a weight matrix W. This yields a new vector z whose elements is put through an activation function f which gives a new vector as input for the next neuron layer, see equation (2.5).

$$a^{(1)} \rightarrow W^{(1)}a^{(1)} = z^{(1)} \rightarrow f(z^{(1)}) = a^{(2)} \rightarrow W^{(2)}a^{(2)} = z^{(2)} \rightarrow f(z^{(2)}) = a^{(3)} \quad (2.5)$$

This new vector then goes through the same operations for the next layer until an output from the ANN is produced.

The number of layers used is referred to as *layer depth*, in practise there are often several layers in a model. This branch of machine learning is called Deep Learning and is centred around Artificial Neural Networks (ANN) of greater layer depth. Information travels from input neurons to output neurons, passing through multiple processing layers composed of linear, and non linear transformations (Goodfellow et al., 2016). For each new layer used, a weight matrix is added and so the equation (2.4) operates over and over again.

2.2 Convolutional Neural Networks

As described by Goodfellow et al. (2016) CNN (LeCun et al., 1998) is a type of neural network that processes data with a grid-like topology, e.g image data that can be viewed as a 2D grid of pixels. The network gets its name from the mathematical linear operation convolution. While regular neural networks use regular matrix multiplication in its layers, CNNs make use of convolution instead.

2.2.1 Convolution

Convolution is an operation on two functions of a real-valued argument (see equation 2.6). Goodfellow et al. (2016) state that in CNN terminology the first argument, x, is known as the input, the second argument, w, is known as the kernel and lastly the output is referred to as the feature map. Figure 2.3 depicts an example of 2D-convolution where the output is restricted to positions where the kernel lies within the input. This implies that the convolution is "valid" according to Goodfellow et al. (2016).

$$s(t) = \sum_{a=1}^{t} x(a)w(t - a) \tag{2.6}$$

Goodfellow et al. (2016) also state that convolution enables improvement in systems with help from three concepts: *sparse interactions*, *parameter sharing* and *equivariant representations*

In regular network layers every input unit interacts with every output unit, due to the use of matrix multiplication. However, the interaction between the input units and output units is usually sparse in CNNs. The interaction can be minimized by making the kernel smaller than the input. According to Goodfellow et al. (2016) *sparse interactions* come with several advantages. Imagine image processing where the image input has millions of pixels. Essential features can be detected with kernels that take up fewer pixels. This leads to the need of fewer parameters, which decreases the memory requirements of the model and also improves its statistical efficiency. Furthermore, less operations are required for computing the output. This gives us a significant improvement in efficiency.

Figure 2.3: Figure based on gure 9.1 from Goodfellow et al. (2016). The gure shows how the convolution operation works.

Parameter sharing is used in a CNN to reduce the number of parameters in the network. The idea is to use the same type of parameters for more than one function in a model. Parameter sharing only occurs in a CNN due to the convolution operator, where each element in the kernel is used at each element of the input except for the outer pixels if no padding is used, see gure 2.3. In a traditional network, however, each element in a weight matrix is used exactly once when multiplied by an input, meaning that no parameter sharing takes place. While the run time of the model remains una ected, the storage requirements reduces signi cantly due to parameter sharing. Thus, in terms of memory requirements convolution is preferable over matrix multiplication. The layers in a CNN inherit a property called equivariance , caused by parameter sharing. The meaning behind the equivariance property is that when there is a change in the input, the same change will happen on the output (Goodfellow et al., 2016).

2.2.2 Max pooling

Max pooling is a function that helps preventing over tting by reducing the number of parameters and operations for computing the output. The function does this by taking the maximum value in a rectangular area by applying a lter on each region (Goodfellow et al., 2016), see gure 2.4 where the input is a 4x4 matrix with a 2x2 lter.

Figure 2.4: Image depicting max pooling. The value of the pooled cells are the maximum of 2x2 cells in the initial layer.

2.3 Recurrent Neural Networks

A simple RNN can be thought of as a network that processes a sequence on the vector $x^{(t)}$ where t is the time index ranging from 1 to (Goodfellow et al., 2016). Therefore some sort of recursion must be present in order to include previous time steps in such a way that the output is dependent on the whole sequence. As Graves (2008) previously have mentioned, if the condition of ANN is relaxed, and allow cyclical connections, the result would yield a RNN. This concept is illustrated in gure 2.5 where the black box indicates a delay for each and every time step.

Figure 2.5: A simple model of a recurrent hidden layer h where the black box indicates a delay for each and every time step.

Mathematically speaking, this recurrent hidden layer utilizes equation (2.7) to de ne the next hidden state vector, thus making h depend on its predecessor and the current input. This equation is an adaptation from what Goodfellow et al. (2016) uses when describing the recursion in RNN.

$$h_t = (Wx_t + Uh_{t-1}) \tag{2.7}$$

Furthermore, looking back at gure 2.5 there is no good visual cue for how the RNN over time stores each hidden layer's information from previous time step. Therefore gure 2.6 presents a more elaborated version of gure 2.5 where output $y^{(t)}$ is also

present. This gure presents the unfolding of the network over time, and visualizes how the current hidden states contain information from previous time step with the help of geometrical shapes. Also, the same gure illustrates the concept of a many-to-one network where a sequence of data is classi ed by the last output, for instance, sentiment classi cation (Karpathy, 2015). Therefore the dotted outputs are discarded.

Figure 2.6: A RNN unfolding over time where each word is a input at di erent time steps. The colored geometrical shapes mimics the conservation of contextual information.

This idea of conserving contextual information between input and output is what makes a RNN bene cial over other archetypes (Graves, 2008). Though, there are unfortunately limitations for how long the context can be accessed. The problem here lies in the reoccurring inputs in the hidden layer which eventually make the network's output, either to blow up or decay as it recursively cycles around the connections (Graves, 2008). This phenomenon is brought up by Hochreiter et al. (2001) and is entitled the vanishing gradients problem, visualized by gure 2.7.

Figure 2.7: Adopted from Graves (2008), the gure visualizes with shading that early context have less impact in later time step. Thus the rst input have no impact on the output since new input overwrite the activation of hidden unit.

With the introduction of vanishing gradients it's a quite natural course of action to present a solution in form of a new type of RNN architecture named Long Short-Term Memory.

2.3.1 Long Short-Term Memory

Hochreiter and Schmidhuber (1997) introduced the LSTM unit, a RNN architecture, as a way of learning long time dependencies and thus reducing the vanishing gradient problem. Since then, the unit has been slightly modified. The following implementation is taken from Chung et al. (2014).

The LSTM unit is composed of three gates: input i, forget f, output o and two memory cells: existing memory cell c, and new memory cell content $\tilde{c}$. The idea is that by using gates to control the flow of information in the unit, it can decide whether to keep or discard certain information in the memory cells for each layer at time step t. See figure 2.8 for an illustration.

The memory cell is computed by equation (2.8), where it is updated by partially forgetting the current memory state whilst also adding new memory content. Equation (2.9) describes the computation for the new memory content, where W_c, U_c are weight matrices and $\tanh$ is the activation function.

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t \tag{2.8}$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1}) \tag{2.9}$$

The input gate regulates the amount of new memory content that is added to the memory cell. See equation (2.10). Moreover, the forget gate in equation (2.11) controls how much the existing memory is forgotten. W, U, V are weight matrices and σ is the activation function.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1}) \tag{2.10}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1}) \tag{2.11}$$

The output of the LSTM unit, h, can be described as in equation (2.12), where o_t denotes the output gate seen in equation (2.13). The output gate regulates the amount of memory that is exposed to other units in the network.

$$h_t = o_t \tanh(c_t) \tag{2.12}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t) \tag{2.13}$$

Figure 2.8: Figure of a LSTM, taken with permission from Chung et al. (2014).

2.3.2 Gated Recurrent Unit

Cho et al. (2014) proposed a type of unit called a Gated Recurrent Unit (GRU), which shares a similar architecture to the LSTM unit. Both units regulate the flow of information with the use of gates. However, the GRU lacks an output gate which means that unlike the LSTM unit, the amount of exposed memory to other units is not regulated at all. Thus, at each time step the GRU fully exposes its memory content (Chung et al., 2015).

Studies comparing the LSTM unit and the GRU have shown that both units perform similarly (Chung et al., 2014).

The following implementation of the GRU is taken from Chung et al. (2014). Like the LSTM uses its three gates to update its memory cell, the GRU updates its memory content, h, by using the reset, r, and update, z, gate. See figure 2.9 for an illustration of the unit. The memory content is a linear interpolation between the previous state of the unit and the new candidate memory content, $\tilde{h}$.

The memory content is controlled by the update gate, as it determines precisely how much the content is updated. See equation (2.14), where $\tilde{h}$ is the new candidate memory content.

$$h_t = (1 - z_t)h_t + z_t\tilde{h}_t \tag{2.14}$$

The candidate memory content is computed by using the reset gate, which allows the unit to forget its previous state to a degree which is determined by the reset gate, as shown in equation (2.15), where W, U are matrices and $\odot$ denotes element-wise multiplication between the reset vector and the previous state of the unit, h_{t-1}.

$$\tilde{h}_t = \tanh(Wx_t + U(r_t \odot h_{t-1})) \tag{2.15}$$

The reset gate is computed by equation (2.16). W_r, U_r are matrices and σ is a sigmoid function.

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{2.16}$$

Similar to the reset gate, the update gate is computed by equation (2.17). W_z, U_z are matrices and σ is a sigmoid function.

$$z_t = \quad (W_z x_t + U_z h_{t\ 1}) \tag{2.17}$$

Figure 2.9: Figure of a GRU, taken with permission from Chung et al. (2014).

2.4 Training

In order for an ANN to generate desirable output, the weights must adopt proper values. This is accomplished by training the network. There are two common types of training for di erent tasks: supervised learningand unsupervised learning (Goodfellow et al., 2016). This thesis will focus on supervised learning for a binary classi cation task.

The goal of supervised learning is to nd the set of weights which optimizes the model to t a given set of input and output data. A dataset contains a list of datapoints consisting of input data and the corresponding output data of an unknown model. This dataset is randomly partitioned into three subsets: thetraining set, the cross validation setand the test set The training set is the set of points to which the model is optimized during the training process. The cross validation set is used to evaluate how well the trained model generalizes, that is, how it performs on data on which it has not been trained. By analyzing the cost function, see section 2.4.1 below on the cross evaluation set with respect to hyperparameters, these can be adjusted to build a model which generalizes better. Finally, the test set is the set on which the built and trained model is evaluated, and this will return the reported score. The test data should not in any way in uence the network optimization process in order to ensure a fair and accurate estimation of model performance on general data.

Since a dataset is an essential tool for training a classi er, the end result is heavily dependent on the quantity and quality of it. A general guideline for acceptable results is to have a dataset of at least 5000 datapoints (Goodfellow et al., 2016), a demand that the datasets in this paper meets with good margin. A common de nition of data quality is correct representation of the real-world construct to which the data refers (Roebuck, 2011). As such, a dataset must not contain a high proportion of noisy datapoints, such as incorrectly labeled or inconsistently attributed.

2.4.1 Cost function

In order to fit a model to a training set, a quantitative measurement network output error with respect to the weights must be defined. This is achieved with a so called cost function, $J(W)$. Different cost functions may be used for different tasks, but they are in general defined to strictly grow as the outputs of a model deviate from the target outputs given a set of data. (Elkan, 2001) Therefore, finding the weights that minimize this cost function will yield a model which performs well on the set on which it is trained. For the task of binary classification, the binary cross entropy loss function is commonly used. The function is stated in equation (2.18) where $t^{(i)} \in \{[0,1],[1,0]\}$ denotes the true label, and $y^{(i)} \in \{[0,1],[1,0]\}$ is the model output for data point m. $[1,0]$ and $[0,1]$ are the numerical representations of the two distinct labels.

$$J(W) = \frac{1}{m}\sum_{i=1}^{m} t^{(i)} \log y^{(i)} + (1 - t^{(i)}) \log 1 - y^{(i)} \tag{2.18}$$

2.4.2 Gradient Descent

Minimizing the cost function of a large ANN is a problem which is too complex for an analytic approach. Instead a numerical approach must be used. The method of Gradient Descent is used to find a local minimum of an arbitrarily variate function.

Ruder (2016) states that there are three different variants of gradient descent: Batch Gradient Descent (BGD), Stochastic Gradient Descent (SGD) and Mini-batch Gradient Descent (MbGB). The reason behind these modifications is to minimize the long computation time for batch gradient descent since the three methods depend on different amount of data used for each step in the algorithm. This is one of three difficulties that White and Ligomenides (1993) mention in their paper where they describe that a large neural network is prune to slow training and a more complex error surface. Even though this project isn't directly utilizing any of these, it is important to understand the basics and their challenges. This project uses the optimized gradient descent algorithm Adaptive Moment Estimation (Adam) which is later described in section 2.4.2.1.

For the GD algorithm, the initial weights are randomized and then updated iteratively by taking steps in the opposite direction of the gradient of $J(W)$ evaluated on the current weights, see the update rule equation (2.19) where i denotes iteration. The size of the step taken is the absolute value of $J(W)$ multiplied by the learning rate coefficient α.

$$W_{i+1} = W_i - \alpha \nabla J(W_i) \tag{2.19}$$

The gradient of the cost function is the vector defined in the space of weights given by the partial derivatives, see equation (2.20). Calculating the gradient of a cost function in an ANN is a challenge in its own which is achieved using the Backprop-

agation Algorithm, see section 2.4.3 below.

$$\nabla J = \left(\frac{\partial J}{\partial W_1}, \frac{\partial J}{\partial W_2}, \ldots \right)^{\top} \tag{2.20}$$

Because the gradient size $|\nabla J(W)|$ decreases with the rate of change of $J(W)$ in the gradient direction, $W_{i+1} - W_i$ where $J(W)$ changes slowly, that is, at local stationary points. If the weights converge to a stationary point, it will be a minimum. This is because $-\nabla J(W)$ at any given point is oriented in the direction in which the cost function $J(W)$ decreases most rapidly (Weisstein). Unless an iteration yields W that is exactly a local maximum or a saddle point, which is very unlikely, the next iteration will diverge from the larger values of $J(W)$. Thus, only in the close proximity of a local minimum will the weights converge.

The BGD algorithm is simply the algorithm described above where $J(W)$ is the cost function with respect to the entire training set, the full cost function. For SGD, partial cost functions are defined which take into account one randomly chosen data point each per iteration. The full cost function is the sum of all partial cost functions, meaning that the cost function gradient is the sum of all partial gradients. Despite not necessarily yielding sets of weights W closer to the cost function's minimum every step, the method has been shown to almost surely converge to a local minimum nevertheless (Bottou, 1998). This method is advantageous in that fewer calculations are needed for each step which may be directed toward the local minimum, and thus faster convergences are achievable. MbGD is similar to SGD, but instead of only taking one data point into account for every iteration, a batch of n datapoints is used. This yields steps with less variance in direction than SGD, increasing the probability that each step is headed in the right direction while also taking less time to calculate than BGD.

2.4.2.1 Adaptive moment estimation

The vanilla gradient descent algorithm presented above is not without its flaws. Ruder (2016) pointed out that choosing an appropriate learning rate for the algorithm is a difficult task where smaller values will slow down the process whereas larger values might hinder convergence. The algorithm also tends to converge to sub-optimal local minima rather than the global minimum. Additionally, Dauphin et al. (2014) argues that in the proximity of stationary points the gradient tends to take on small values in all directions, causing slow progression of the search. They have also shown that saddle points in particular are very prevalent in higher dimensional problems.

To overcome said challenges, numerous gradient descent optimization algorithms have been developed with more sophisticated update rules based on the vanilla. Among these is the *Adaptive Moment Estimation* algorithm (Adam) proposed by Kingma and Ba (2014) which aims to improve SGD and MbGD firstly by setting a step size invariant to the gradient and secondly having moment from the previous iteration preserved in the next. The invariant step size ensures that the search will

Figure 2.10: Figure demonstrating optimization paths taken by the vanilla gradient descent and Adam algorithm, both starting from the same initial point. The blue line represents GD, while the red line represents Adam. Note that Adam in this gure converges to the global minimum whereas vanilla GD converges to a local due to the moment property of Adam.

not slow down close to saddle points, and in conjunction with moment it also allows for skipping past sub-optimal minima as demonstrated in figure 2.10. By utilizing moment, that is setting the new step direction to a weighted sum of the current gradient and the last step direction, the problem introduced with stochastic optimization of not stepping in the right direction is diminished by taking into account the directions of partial gradients in all iterations. Adam also implements an adaptive learning rate for different weights by having the influence of partial gradients exponentially vanish which gives sparse features greater impact on the search path. This is useful in our problem where many words in our dataset that may have impact on the classification are infrequent, see section 3.1.

2.4.3 Backpropagation

To calculate the gradient used in SGD, previously mentioned in section 2.4.2, bacpropagation is utilized. According to Nielsen (2015), the usual way is to propagate datapoints through the neural network to the output layer, compare the output value of the network with the desired output for the said data point using a cost function, and than propagating the error backward through the network to find the error gradient in each of the components in the network.

Since interest lies in finding the gradient, the whole procedure of propagating the error backwards can be done by a long chain of differentiations of the error function with respect to each of the components in the model using Hamiltonian/Lagrangian formalism (Le Cun et al., 1988).

Hecht-Nielsen (1989) describes the mathematics behind the backpropagation algorithm can be described with following: The first step is to differentiate the cost function $J(W)$ with respect to the output y_j for a neuron in the output layer. Which in the general case are annotated as equation 2.21.

$$\frac{\partial J}{\partial y_j} \tag{2.21}$$

The output y_j is also a function of the input to said layer z_{ji}, i.e equation 2.22.

$$y_j = g(z_{ji}) \tag{2.22}$$

With z_j being the scalar product of the weights W_{ji} and the outputs of the layer below (i.e. $z_{ji} = W_{ji}^T y_i$), which can also be viewed as a function. The chain rule of differentiation can be utilized in finding the gradients of the weights in the network. For a network with one neuron in each layer, the gradients can be calculated as equation 2.23.

$$\frac{\partial J}{\partial z_{ji}} = \frac{\partial J}{\partial y_j} \frac{y_j}{z_{ji}} \tag{2.23}$$

Using the chain rule, and the already calculated differentiation of the cost function with respect for the output, the differentiation of the error with respect to the input z_{ji} to the output neuron y_j are the said calculation, performed above. This results is

later used to di erentiate the cost function with respect to the output of the layer below (y_i). See equation 2.24.

$$\frac{\partial J}{\partial y_i} = \frac{\partial J}{\partial z_j} \frac{z_j}{y_i} \tag{2.24}$$

By inserting equation 2.25 in equation 2.24 equation 2.26 is obtained.

$$z_{ji} = W_{ji}^T y_i = \sum_i W_{ji} y_i \quad \Rightarrow \quad \frac{z_j}{y_i} = W_{ji} \tag{2.25}$$

$$\frac{\partial J}{\partial y_i} = \frac{\partial J}{\partial z_j} \frac{z_j}{y_i} = \frac{\partial J}{\partial z_j} W_{ji} \tag{2.26}$$

Now, to calculate the error gradient with respect to the weight to the layer W_{ji} equation 2.27 is used.

$$\frac{\partial J}{\partial W_{ji}} = \frac{\partial J}{\partial z_j} \frac{z_j}{W_{ji}} \tag{2.27}$$

And then again equation 2.28.

$$z_j = W_{ji}^T y_i = \sum_i W_{ji} y_i \quad \Rightarrow \quad \frac{z_j}{W_{ji}} = y_i \tag{2.28}$$

Inserting equation 2.28 into equation 2.27 yields equation 2.29.

$$\frac{\partial J}{\partial W_{ji}} = \frac{\partial J}{\partial z_j} \frac{z_j}{W_{ji}} = \frac{\partial J}{\partial z_j} y_i \tag{2.29}$$

The calculations for a network with more than one neurons in each layer, the di erentiations are almost the same, but since the output of one neuron can now a ect multiple neurons in the layer above , all of these has to be taken into consideration. Hence a sum has to be added in the step where the error function is di erentiated with respect to the layer below the output (i.e. $\frac{\partial J}{\partial y}$, see equation 2.24). In other words, the error gradient with respect to the output from a layer below is equal to the sum of each error gradient from the layer above that are connected to the neuron in equation 2.30

$$\frac{\partial J}{\partial y_i} = \sum_j \frac{\partial J}{\partial z_j} \frac{z_{ji}}{y_i} = \sum_j \frac{\partial J}{\partial z_j} W_{ji} \tag{2.30}$$

The error gradient $\frac{\partial J}{\partial z_j}$ is calculated the same way as in equation 2.23. And the weight gradient $\frac{\partial J}{\partial W_{ji}}$ is calculated the same way according to equation 2.27

2.5 Over tting

The training process as described above in section 2.4, is essentially the adaptation of the model to a certain set of datapoints. As the training proceeds, the model will t better and better to this speci c set. If the model is designed properly, this will capture the general trend among this type of datapoints. If the training proceeds

for too long, while using a too complex model relative to the number of datapoints however, the t to this particular set will be too good in the sense that the model will start to remember this speci c data. Rather than just learning the general, the classi er captures even small deviations from normal tendencies. This leads to a loss of generality where the classi er gets overly sensitive to noise in the data, see gure 2.11.

Figure 2.11: The picture shows two di erent classi ers, black and purple curves, for predicting dots and crosses based on position. The black curve depicts the overall trend among the datapoints in the training set, which is preferable for classifying future data. The purple curve represents an overly complex model which is perfectly adjusted to the training data but is unfavorable for classifying future datapoints. In the latter case the model is said to be over tted.

This e ect is called over tting and means that the training has produced an overly complex model. Having too many parameters relative to the number of datapoints is another situation where the model might tend to over t. To minimize the e ect of over tting, a few techniques are available. Using cross validation during the training phase will make it possible to monitor how well the model generalizes, which can be used with theearly stoppingtechnique. Early stopping basically means that the training is completed as soon as the cross evaluation cost function start to increase after passing its global minimum, which can be seen in 2.12.

Figure 2.12: The picture describes the effect of overfitting. After an initial decrease of both the Test set's and Cross validation set's Cost function, the latter tends to increase due to loss of generality in the classifier. The early stopping method can be used to prevent this, where the training ends at the minima of the Cost function.

Regularization refers to a technique where a weighted Regularization term, $R(W)$, is added to the cost function, in order to dampen the model's complexity. The Regularization term can be defined in different ways. One of the simplest ways is to use a sum of the squared weights in the weight matrix (Bishop, 2006) (page 257). With this term, the problem thus becomes to minimize the cost function with the the added weighted regularization term, see equation 2.31.

$$\min_W J(W) + R(W) = \min_W J(W) + \sum_i |w_i|^2 \tag{2.31}$$

The effect of this is that the model still is fitted to the training data, but with the constraint of limiting the complexity of the function that the weight matrix represents. In simpler terms, the goal is to find the simplest function that solves the problem. The strength of the regularization term is controlled through the parameter , which thus decides the degree of this regularization effect.

Another method that is commonly used to avoid overfitting is Dropout. Srivastava et al. (2014) describes this method to work in such a way that it randomly turns off units, both in the visible and the hidden layer, to avoid the model learning the dataset by heart.

2.6 Scoring methods

Multiple methods exist for evaluating binary classifiers. They are usually based on the use of datapoints that belong to one of four categories: True positive (TP), True negative (TN), False positive (FP) and False negative (FN). These subsets describe if the classification was correct or not (i.e. true or false) and what the classification

was (positive or negative), (Fawcett, 2006). The notion of positive is arbitrary, but is usually defined to mean the relevant category. In this report, positive means the tweets belonging to the sarcastic class. This subset of points can be used to describe the evaluation methods as follows in the following subsections. Two of the most common methods within the field of sarcasm detection will be described here: Accuracy (ACC) and F1-score (F1), though other metrics have been used as well (Joshi et al., 2016b). F1 is by far the most common choice. The reason is due to the fact that the accuracy measure tend to be problematic, especially on unbalanced datasets (Parker, 2013). Both these methods outputs a number between one and zero, where a higher number signals classification with greater confidence.

2.6.1 Accuracy

ACC is in a sense a measure of the absolute accuracy. It measures the proportion of correctly classified datapoints to the total number of datapoints. In mathematical notation, this can be written as equation 2.32.

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.32}$$

The reason that the ACC measure might be misleading is that it tends to give an unreasonably good score by favoring the majority class in a very imbalanced set. For example, when diagnosing patients for a rare disease, a doctor might blindly give every single patient a negative diagnose. This will give a very high accuracy even though the doctor made a poor job. For this reason, better metrics exist for imbalanced datasets.

2.6.2 F1-score

F1 presents a better metric than ACC in the case of imbalanced sets, since it takes into account the disproportional distribution of the two classes. To understand F1, the two statistics Precision and Recall need to be presented. Precision and recall are mathematically defined as (Powers, 2011). See equation 2.33 and 2.34.

$$Precision = \frac{TP}{TP+FP} \tag{2.33}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.34}$$

In terms of sarcasm classification, Precision is the proportion of the tweets classified as sarcastic that actually are sarcastic. Recall is the proportion of the sarcastic tweets that was correctly classified. F1 is defined as the harmonic mean of precision and recall, which means that F1 can be described as equation 2.35.

$$F1 = 2\,\frac{Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{2TP+FP+FN} \tag{2.35}$$

The metric F1 punishes tendencies to overly classify towards the majority class, which is preferable especially when using unbalanced datasets.

2.7 Word embeddings

One of the first problems within machine learning, and data analysis in general is, to find a suitable numerical representation of the collected data samples. In the case of text classification problems, which sarcasm detection is a part of, the problem is to find a suitable numerical representation of the text that is to be classified.

One way of doing this is to represent every word in the text as a real value vector in a multidimensional space. This is called word embedding.

In a word embedding model, every word is represented as a vector in D dimensional space, where D is a relative small number (usually 50 - 1000). Words with similar semantic meanings will be grouped together with similar words, which means that their vectors will be similar in the sense that their coordinates are close to each other in the vector space.

The term word embedding was introduced by Bengio et al. (2003). Bengio also proposed that such a model shall be trained using a neural network like architecture of function decomposition to train a such a model. The paper proposes that this is way to achieve the objective of finding a good model $f(w_t, ..., w_{t-n+1}) = \hat{P}(w_t | w_{t-1})$ such that it gave a good out of sample likelihood.

The technique was not utilized as frequently in until recently, with the development of the Word2Vec training algorithm in 2013 by Mikolov et al., at Google.

Another word embedding training algorithm was presented in 2014 by Pennington et al. at Stanford University called the GloVe model.

3
Method

The datasets used contained labeled tweets. Before they were used for training the models they needed to be preprocessed. This entails transforming them into a numerical representation, and removing any labeling information, for instance the sarcasm hashtag.

To proceed with the training step, we needed to design the networks and their properties. The di erent properties we could have controlled are commonly called hyperparameters These in uence a network's ability to learn, which means that nding an e ective con guration of hyperparameters could be an important step for maximizing performance. To implement the network we have chosen the open source software libraryTensor ow developed by Google.

To examine our trained network's functionality, we have used a couple of methods. Most importantly, we were concerned with whether the network looks at a tweet as a whole, which means taking into account features such as the order of the words and whether the tweets contain words of opposing sentiment. We have also examined the in uence of randomness during the training, that is whether a large fraction of tweets tend to di er in classi cation from a training to the next. Another point of interest is what type of data the network trained on a single dataset can handle, speci cally whether it can generalize to one of the other datasets.

In order to obtain a baseline of human performance we performed an online survey using volunteer Chalmers students.

The following sections describe these concepts in detail, starting with details regarding datasets, survey for human ability to classify sarcasm, training, model designs, training and testing proposed models and implementation of the networks.

3.1 Datasets

In order to compare our results with state-of-the-art we chose the dataset created by Pta£ek et al. (2014), which as previously mentioned was also used by Poria et al. (2016), see section 1.3. The datasets are comprised of 100,000 tweets each: one balanced and one imbalanced. The balanced set contains 50,000 sarcastic and an equal amount of non sarcastic tweets, and the imbalanced contains 75,000 non sarcastic and 25,000 sarcastic tweets. The #sarcasm hashtag was used to label the

sarcastic tweets. As we were only able to acquire the tweet ID, we had to download the datasets from Twitter using their API. Some tweets were no longer available, consequently our datasets are smaller, around 75,000 tweets each.

Another dataset that we used was obtained from The Sarcasm Detector, which contained ca 150,000 sarcastic and ca 330,000 non sarcastic tweets. They collected their data similarly to Ptá£ek et al. (2014), where they gathered the sarcastic tweets with the use of the #sarcasm hashtag and non sarcastic tweets under a three week period. The di erence from the dataset we obtained from Ptá£ek et al. (2014) is that the dataset from The Sarcasm Detector contained raw text, which meant that we had to give them separate tweet ID's and were not able to obtain the same information as to when we extracted the dataset of Ptá£ek et al. (2014) from Twitter's API. The bene t of the dataset from The Sarcasm Detector is that no tweets were lost as compared to when collecting the dataset of Ptá£ek et al. (2014).

3.1.1 Data preprocessing

This project utilized two di erent types of preprocessing, *strict* and *non-strict*. The two di erent types determine to which extent the datasets are cleaned. The main cleaning method used in this project for all the networks is the non-strict method, whereas the strict method is used for special purposes which will be described below.

The *strict* cleaning method removed all tweets if they began with a user mention (@), a retweet (RT) or if they contained any urls. We also decided to remove duplicate tweets and tweets generated by bots. Lastly all tweets with a length smaller than three, after the tags were removed, were disposed of. The *non-strict* cleaning method keeps all tweets as they are. In both cases the hashtag #sarcasm is completely removed. Additionally, we included an option on how we want to handle tags such as other hashtags, user mentions and urls. This was done in order to remove any information that could bias the classi er, but at the same time save the syntactic signi cance. See table 3.1 for an overview of tags and no-tags.

Table 3.1: Table describing the handling of tags

Type	Hashtags	User mentions	Url
Tags	<hashtag>	<user>	<url>
No-tags			

Since we wanted to compare our results with state-of-the-art, we wanted to clean our data equivalently to them. Their cleaning method represents the non-strict one. The strict preprocessing method was used for the survey covered in section 3.6 below.

Both Poria et al. (2016) and Ptá£ek et al. (2014) use the non-strict cleaning method. The di erence is how they handle the tags in the tweets. Poria et al. (2016) do not replace hashtags, user mentions and urls with any tags, see table 3.1. They remove them from the tweet entirely. Ptá£ek et al. (2014) on the other hand replace the tags as depicted in table 3.1. We also wanted to evaluate how tag handling a ected

our results, i.e if including or removing tags in our dataset would yield better results.

The next step was to split the tweets into words, or *tokens*. The tokens include non-word strings that convey meaning, such as emojis. To achieve this the NLTK library was used, which provides the ability to tokenize informal language. The tokenized words where then counted and the total occurrences within the corpora tallied. The most commonn words where then used to build a dictionary, mapping each word to an unique integer value, called the *vocabulary*. The uncommon words that didn't fit were given the same placeholder value. When feeding the tweets to the network we made use of *word embeddings* see section 2.7.

3.2 Description of used models

For the task at hand, three different models have been implemented and tested. Two different RNNs, one using LSTM-cells and the other GRU-cells with 256 hidden units each (see 3.2), as well as a CNN model with a filter size of 128 and a filter width of 3, 4 and 5. See table 3.2 for full model description. Each has the input layer connected to an embedding layer that is connected to the specific main units of the networks. Depending on which network, it is either the LSTM, GRU or CNN layer, which in turn is connected to the fully connected output layer with a softmax activation function.

Table 3.2: Overview of the used models.

Model	Number of parameters
LSTM	hidden memory unit = 256
GRU	hidden memory unit = 256
CNN	filter size = 128 + filter width = 3,4,5

Figure 3.1: The implemented LSTM/GRU network using 256 LSTM/GRU cells, 20,000 vocabulary size, word embeddings of size 200, and 75 time steps.

3.3 Hyperparameter optimization

A part of optimizing the network performance sometimes lies in finding the optimal configuration of the hyperparameters. Some networks are very sensitive to these parameters and finding them is crucial. In our case we were interested in conducting experiments with several distinct values ranging from lowest to highest specified value, see table 3.3.

Table 3.3: Hyperparameters used while conducting the hyperparameter search ranging from the lowest to the highest value.

Parameter	Lowest value	Highest value
Dropout	0.4	0.8
Learning rate	0.0005	0.0015
Weight decay	0.01	0.06

To do this, scalar values of the hyperparameters were selected according to a random uniform distribution from table 3.3 values before training the network, and the training process was repeated iteratively with randomly selected values. Our approach to the problem would not necessarily yield the best possible hyperparameters, but should provide some decent ones or even better, prove that the model was not that sensitive to changing the hyperparameters.

In order to evaluate whether or not the choice of hyperparameters had a significant effect on network performances, each iteration was logged along with the results for further analysis.

3.4 Training and testing models

Since a method of preprocessing the datasets has been decided upon, and a suitable choice of hyperparameters has been found, each model was trained and evaluated on each of the datasets. After a model has been trained on a dataset, it is also evaluated on test partitions of the data in the other datasets. This is to yield additional results for better evaluation whether the trained model generalizes well.

3.5 Analysis of network functionality

To examine the functionality of the networks, we have conducted a few experiments. An interesting question to be answered was whether the network actually base its decisions on features resembling sarcastic intent. This is not a trivial question even though the results might look good at a first glance. The reason is that the network might pick up on other trends for sarcastic tweets, such as a high frequency of typically sarcastic words, length of the tweet and excessive use of hashtags. Since one might yield decent performance just by looking at word content, the results of the network were compared to those of a word frequency model for classification.

Sarcasm may also be highly dependent on sequential information, with strong changes in sentiment as possible indicators of sarcasm. Figuring out if the model takes sequence into account was made by randomly scrambling the order of the words in a tweet, and looking at changes in the output. The following experiments were conducted to yield results that helped us answer these questions. Each of the experiments were limited to only being conducted with the network and dataset which yielded the highest F1-score on the test partition in the initial tests. This is referred to as the *base case* in the experiments. Furthermore, said network was trained and evaluated 30 times on the dataset to measure mean and standard deviation of the F1-score with respect to randomly initialized weights, hyperparameters and batches for ADAM optimization. A small standard deviation would indicate that the experimental procedures are mainly responsible for deviations in scores compared to the base case.

3.5.1 Bag-of-words model

If a decent chunk of tweets of one label or the other happen to contain certain words, then there is a chance that the network picks up these as *key features* for labeling. To find out whether this was the case, we performed a word frequency analysis of the tweets in our dataset, by using a basic *bag-of-words model*. First, a glossary was created from all individual words in the dataset. For each of these words we counted the number of times they belong to a sarcastic tweet (n_s) and a non sarcastic tweet (n_u). If a certain word occurred more often in the sarcastic tweets, it was deemed to be a sarcastic word. We scaled how sarcastic a word was as the relative frequency of occurrence (n_f) between the two classes according to equation 3.1.

$$n_f = \frac{n_s - n_u}{n_s + n_u} \tag{3.1}$$

n_f is measured on a scale in the interval $[-1, 1]$ where 1 corresponds to a sarcastic word, and -1 to a non sarcastic word. When classifying a certain tweet, the model added up the individual words' n_f value. If the sum was greater than 0, the word was labeled as sarcastic. The F1-score of the output was then compared to the network score. From these results, a comparison between neural network performance and a simple rule based model was made.

3.5.2 Word scrambling

For this experiment, the test partition was preprocessed in such a way that the words in each tweet occur in a random order. This was done by randomly shuffling the words in each tweet. We compared the performances of models trained on a dataset and then tested on both the original test partition and the word scrambled version of it. This allowed us to draw conclusions on whether the network takes into account the sequential order of words in the tweets.

3.5.3 Trouble makers

All tweets are given unique ID's which remain set during all operations. Exploiting these, we could after several training sessions of the base case determine if certain tweets tend to be incorrectly classi ed by a trained network regardless of the initial weights. Data points incorrectly classi ed by 30 classi ers were informally de ned as *trouble makers* for further analysis. Identifying the trouble makers helped us analyze the network functionality by manual inspection of features that may bias the network output. In order to draw conclusions, these features were also inspected in the set of tweets correctly labeled in every run. Additionally, we conducted an experiment using the trouble makers to potentially improve the network performance on the dataset. This was done by training the base case network initially only on these tweets in order to subsequently train it on the rest of the training data using the yielded weights for initialization.

In order to nd the trouble makers on our datasets, the network training process was run 30 times. For each run, the network predicted the labels of all data points in the training set. Each tweet and the number of times it was correctly evaluated was stored in a table along with the true label of the tweet. From this table, a histogram of the fraction of correctly labeled data was constructed. A large encapsulated area at the lower end of the histogram indicates a large amount of trouble makers in the dataset.

3.5.4 Impact of word embeddings

We also wanted to evaluate what impact word embeddings have on our results, by conducting an experiment where we tested random initializing our word embeddings instead of using the initialization by the GLoVe model. The obtained results will be compared to the results of Poria et al. (2016) who conducted a similar experiment. See section 4.2.4.

3.6 Human ability to detect sarcasm

As previously mentioned the survey's purpose was to provide a human baseline for the classi cation of tweets as sarcastic or non sarcastic.

The survey was performed in the form of a quiz, hosted as a web application and shared among students at Chalmers University of Technology. A batch of three random samples from the dataset were displayed at a time. The displayed samples each had two buttons for labeling the sentence sarcastic or not. When the third and last sample had been classi ed, a new batch of three samples was presented together with a table of all the previously mentioned scoring metrics (see section 2.6). The participants were not exposed to any samples of the collected data before the survey, to prevent biased judgment.

Each individual's F1-score was recorded, and the mean of all participants' score

was taken as a measure of human ability. To assure a level of certainty concerning an individuals performance, we only included contributions where a participant completed at least ve rounds of samples, that is at least 15 classi cations.

The strict cleaning method was implemented for use in human sarcasm detection, and to test the hypothesis that certain tweets, such as re-tweets, bot posts, and links, contain little information describing sentiment. Consequently such tweets were hypothesized to be boring to read for humans, so as not to discourage participation in the survey, it also employed the strict cleaning method. In order to compare our model with human performance in the classi cation task, we also evaluated the strictly cleaned data on our RNN with LSTM-cells model.

The dataset used in the survey was the balanced dataset from Ptá£ek et al. (2014).

4

Results and discussion

In this chapter we present the yielded results when training and testing the di erent networks. Values and patterns in the results are then discussed and compared to the results of the related works described in section 1.3.

The search for optimal hyperparameters showed that the models were relatively insensitive to di erent settings. F1-scores only varied marginally with randomly generated sets of hyperparameters, initial weights and batch sizes for training. Thus, the reported scores are all produced by networks with the same hyperparameters, which was chosen from a run which yielded promising results, see table 4.1.

Table 4.1: Chosen hyperparameters, used for all conducted training experiments in this work.

Dropout	0.63
Learning rate	0.001
Regularization parameter	0.038

We also present and discuss the results from the experiments for the network analysis. Throughout all the experiments we used the network and dataset which yielded the best result, namely the RNN with LSTM-cells model and the balanced dataset from Ptá£ek et al. (2014), unless stated otherwise. See table 4.2(b) in section 4.1.

Additionally, later in the chapter we present the results from the experiment of humans detecting sarcasm described in section 3.6. Other subjects that are re- ected upon in this chapter are quality of the datasets, obstacles within the project and impact on society.

From now on for readability, the used datasets are assigned with an acronym as follows:

- Ptá£ek et al. (2014) balanced - D1
- Ptá£ek et al. (2014) ratio - D2
- The Sarcasm Detector - D3

4.1 Training and testing the models

The following section presents the results for the training and testing of the three di erent models: RNN with LSTM, RNN with GRU and CNN. Each dataset was partitioned into a training, validation and test set; using 70%, 15% and 15% of the samples respectively.

Each row in the table that is underlined and color coded red (DX) indicates the dataset the network was trained on. The columns, color coded blue (DY), indicate which dataset the network was evaluated (tested) on, yielding the F1-score displayed in the cell. For each network the presented tables include the evaluation of the datasets with and without tags, which is described in section 3.1.1.

Note that the results achieved by cross-dataset evaluation between D1 and D2 are not presented in the table. This is because many tweets are shared between the datasets. Consequently, the network could train on several data points in the train partition of one set which occur in the test partition of the other, causing biased results.

Table 4.2: Result table for the RNN with LSTM-cells where the metrics are represented in F1-score.

(a) Dataset with tags excluded.

	D1	D2	D3
D1	0.816	-	0.678
D2	-	0.67	0.615
D3	0.698	0.475	0.762

(b) Dataset with tags included.

	D1	D2	D3
D1	0.842	-	0.697
D2	-	0.723	0.67
D3	0.717	0.49	0.796

Table 4.3: Result table for the RNN with GRU cells where the metrics are represented in F1-score.

(a) Dataset with tags excluded.

	D1	D2	D3
D1	0.817	-	0.679
D2	-	0.656	0.615
D3	0.687	0.472	0.764

(b) Dataset with tags included.

	D1	D2	D3
D1	0.836	-	0.698
D2	-	0.697	0.664
D3	0.703	0.473	0.786

Table 4.4: Result table for the CNN where the metrics are represented in F1-score.

(a) Dataset with tags excluded.

	D1	D2	D3
D1	0.811	-	0.669
D2	-	0.637	0.594
D3	0.667	0.434	0.756

(b) Dataset with tags included.

	D1	D2	D3
D1	0.824	-	0.694
D2	-	0.676	0.633
D3	0.695	0.466	0.787

In the result tables, it holds in general that the network performance is best on D1, drops with a mean of 0.0492 for D2 and further with a mean of 0.0987 for D3. The highest overall score of 0.842 on D1 was achieved by the RNN with LSTM-cells where the dataset was processed with tags kept. This network is outperformed by Ptáček et al. (2014) using either their SVM or Max Entropy classifier, yielding scores in the range of 88.39 to 94.66. When preprocessing D1 without tags as proposed and done by Poria et al. (2016), the scores drop with a mean of 0.0193 across the networks, indicating better performance with tags remaining. These results are toppled by Poria et al. (2016) by a great margin as they achieved a score of 95.04 with their CNN classifier.

When training on D1 and testing on D3, the test set scores dropped significantly with a decrement of 0.145, 0.138, and 0.13 respectively for the networks. This indicates that our networks did not generalize very well to other datasets for the task of sarcasm classification. However, the same test performed by Poria et al. (2016) generated a greater decrease of score of 0.21, indicating that their results is even less representative of the general performance. The other way around, training on D3 and testing on D1 is visualized in figure 4.1 and table 4.5.

Examining the differences in scores between the networks on the datasets reveals that there is very little change in performance on the data where tags are excluded. On the datasets with tags, the differences are notably higher, where RNN with LSTM scores on average 0.0087 higher than RNN with GRU which in turn is on average 0.0091 higher than CNN. It thus seems as the RNN with GRU-cells model, which had not been used before, does not add anything to this field in terms of performance.

It is clear from the results displayed in the tables of section 4.1 that the performance of the network models varies on each of the datasets. There might be multiple factors that affect the training performances. This includes initial weight initialization, the hyperparameters, dataset distributions and so on. Individually though, the variety of network performance is small.

Figure 4.1: This gure displays all the F1-scores, and the generalizability across datasets when trained on D3 and tested on D1 ranging from Ptá£ek et al. (2014), Joshi et al. (2015), Poria et al. (2016) to our results.

Table 4.5: Comparison of F1-scores with related works on di erent datasets.

Method	D1	D2	D3	D3 => D1
Ptá£ek et al. (2014)	0.947	0.924	0.634	0.530
Joshi et al. (2015)	0.651	0.624	0.608	0.473
Poria et al. (2016) CNN	0.950	0.893	0.880	0.768[1]
Our model	0.824	0.676	0.787	0.717

The more interesting part arises from observing the results from testing a trained model across datasets, i.e. testing a model on a di erent dataset from which it was trained on. One can observe the trend in the test results that the scores will drop signi cantly when testing across datasets. There are many reasons for this drop in testing results. If small enough, for instance, it might be dismissed as oating point error. The reason for the large variance might be related to learning a less applicable model across the datasets.

Since these are di erent datasets, it might not be surprising that the distributions across the datasets may vary. The results in tables above seem to re ect this. Which can be viewed as a good thing, since this could indicate that the networks managed to learn the features of the datasets, and the catches the correlations between them. Another indication of this is that it is quite di cult to train a general model for sarcasm detection, using only a subset of sample data, even when it comes to tweets.

[1] CNN + pretrained features combined with SVM.

4.2 Analysis of network functionality

The following section presents results from the experiments, followed by discussion of how they strengthen or weaken the validity of our models. For the following experiments, the RNN network with LSTM-cells and dataset D1 was used. This base case yielded a mean F1-score of 0.8395 on the test set with a standard deviation of 0.0029 after 30 sample runs.

4.2.1 Bag-of-words model

In this section, the performance of the bag-of-words model is presented. The tables are structured the same way as in section 4.1, which means all datasets are used. The highest performance of 0.809 was measured on D1 when including tags, as in the case of the neural networks. The performance after removing tags varied, with a mean decrease of 0.002, which we consider negligible. This small deviation can be explained by the frequency values seen in table 4.6, and the occurrence of respective tags. The most frequently occurring tag, <hashtag> has a very low frequency, meaning that the presence will have minimal impact on the classi cation by the bag-of-words model. <url> has a greater negative score, but the tag occurs quite infrequently in the datasets. Thus, even though the presence might have a signi cant impact on the classi cation of a single tweet, the few tweets whose label would change would not greatly impact the F1-score on the dataset as a whole.

Table 4.6: Word frequencies of occurrence in the interval [-1,1] for tags in D1, where 1 and -1 represents sarcastic and non sarcastic respectively.

	Tag	Frequency
D1	<hashtag>	0.037
D1	<user>	-0.12
D1	<url>	-0.67

Table 4.7: Result table for the bag-of-words model where the metrics are represented in F1-score.

(a) Dataset with tags excluded.

	D1	D2	D3
D1	0.807	-	0.673
D2	-	0.619	0.672
D3	0.679	0.421	0.743

(b) Dataset with tags included.

	D1	D2	D3
D1	0.809	-	0.67
D2	-	0.64	0.67
D3	0.679	0.419	0.742

The overall agreement between the network and bag-of-words model was examined by comparing the classi cation for each sample between the models. The proportion of agreements was computed to 86 %, a minor indication that the two model share some functionality.

4.2.2 Word scrambling

In this section, the performance when testing on a scrambled test partition is presented. The training and testing was all done within the same dataset, i.e. no cross-dataset performance is used in this section. The dataset notion and color coding follows the same structure as in section 4.1. Scrambled data and Original data describes that the testing was done on the scrambled and original version of the test partitions respectively.

Table 4.8: Result table for the RNN with LSTM-cells trained on the original datasets and evaluated on both the original and word scrambled test partitions.

	Scrambled data	Original data
D1	0.827	0.842
D2	0.683	0.723
D3	0.77	0.796

Table 4.9: Result table for the RNN with GRU-cells trained on the original datasets and evaluated on both the original and word scrambled test partitions.

	Scrambled data	Original data
D1	0.821	0.836
D2	0.639	0.697
D3	0.755	0.786

Table 4.10: Result table for the CNN trained on the original dataset and evaluated on both the original and word scrambled test partitions.

	Scrambled data	Original data
D1	0.800	0.824
D2	0.643	0.676
D3	0.759	0.787

As can be seen from the tables, the performance decreased when testing was done on the scrambled test partitions. The decrease varies between the models and datasets, with a mean score decrease of 0.03 for the nine cases. This equates to a relative decrease of about 4 %. While this indicates that the network captures some type of information related to sequence, this e ect seems quite small.

4.2.3 Trouble makers

The following section contains the results and discussion around the analysis of tweets that was never correctly classi ed during the training of the network, i.e. the following datapoints are from the training partition of D1.

Among the tweets that were correctly classi ed 100% of the time, 51.5% were sarcastic, whereas among those classi ed incorrectly 100% of the time, 45% were sarcastic.

Figure 4.2: Histogram of tweets ranging from always correctly classi ed (0) to tweets that were always wrongfully classi ed (1).

To further display the amount of wrongfully classi ed tweets, the y-axis is limited from 45,000 tweets to 4,500, see gure 4.3.

Figure 4.3: Zoomed y-axis of the previous gure 4.2's histogram of tweets ranging from always correctly classi ed (0) to tweets that were always incorrectly classi ed (1).

Listed in table 4.11 are 10 randomly selected tweets which never were classi ed correctly by any of the 30 networks. By manually inspecting the entire set of collected data, we sought out key features relating to network classi cation. Among the trouble makers which always were classi ed incorrectly, 634 non sarcastic tweets and 78 sarcastic tweets all ended with the token <hashtag>. Tweets featuring the token in the end of the sentence among those that always were classi ed correctly were comprised by 2382 non-sarcastic and 8916 sarcastic. This indicates that tweets ending with the <hashtag> word might be key feature biasing output to the sarcastic label.

Table 4.11: 10 randomly selected troublemaker samples which were never classi ed correctly by the 30 networks trained on D1 with di erent initial weights.

Sample tweet	Predicted label	Actual label
Arrived in Florida all safe and sound except that my suitcase is still in Amsterdam, just my luck of coooooourse :-) :-) :-)	Not sarcastic	Sarcastic
Sooo my tablet and mobile phone are fully loaded I AM READY for <hashtag> <user>	Sarcastic	Not sarcastic
I cant believe My French teacher tickled my friend while she was laughing :O <user>	Not sarcastic	Sarcastic
When the beer and food is so good it distracts you from the shitty service	Sarcastic	Not sarcastic
Print isn't dead. It just smells funny. <hashtag> <hashtag>	Sarcastic	Not sarcastic
It feels like sumthings heating up can i leave wit u <hashtag>	Sarcastic	Not sarcastic
"You need to learn to love yourself, because no one's gonna love you if you don't love yourself"	Sarcastic	Not sarcastic
<user> and I thought we had problems..... The end of big Sam?	Sarcastic	Not sarcastic
I thought we was suppose to be a team I guess not every body for they self right now	Sarcastic	Not sarcastic
After that last interview, I just can't wait to read Dries interview w/ Zap2It tomorrow.	Not sarcastic	Sarcastic

An initial RNN with LSTM-cells was trained only on the trouble makers from D1, and was then trained on the rest of the dataset, including the trouble makers. This network was yielded a F1-score of 0.843 on the test set whereas the non-boosted counterpart yielded 0.842. This marginal increase from the base case is insigni cant as it is well within the standard deviation of 0.0029 from the mean F1-score.

4.2.4 Impact of word embeddings

This experiment evaluated how the initialization of the word embeddings a ected our results. Both training and testing were carried out on partitions from the balanced dataset (D1), one case where the embeddings were initialized randomly and one where we used word vectors from a GLoVe model. As visible in table 4.12 the impact was much smaller than experienced by Poria et al. (2016), who demonstrated a signi cant drop in F1-score when using randomly generated embeddings.

Table 4.12: Results for randomly initialized word embeddings versus pretrained GLoVe or Word2Vec model compared with Poria et al. (2016).

Model	F1-score
Our model with random embeddings	0.8220
Our model with GLoVe embeddings	0.8420
Poria et al. (2016) with random embeddings	0.8623
Poria et al. (2016) with Word2Vec embeddings	0.9771

These numbers may indicate that the GLoVe model might have been ill-suited for our task unlike the Word2Vec model. Poria et al. (2016) results indicates that Word2Vec word embeddings was a better suit since the increase in performance is greater compared to our model where GLoVe was used.

4.3 Human ability to detect sarcasm

In the survey, a total of 4086 classi cations were collected from 219 participants. After removing the ones with 15 or fewer classi cations, 3768 classi cation and 110 participants remained. A histogram with the number of participants with a certain F1-score can be seen in gure 4.4, where the mean F1-score was 0.791 with a standard deviation of 0.149. Our LSTM implementation trained and tested on the same dataset, achieved an F1-score of 0.775.

We further investigated the usefulness of the dataset for this task. A high agreement between human classi cations would indicate a dataset well-suited for this experiment. The opposite case would instead result in values close to 50 % agreement, representing zero correlation and bad data. Among the 3768 classi cations, 202 tweets had been classi ed by at least two di erent users. In 78.3% of these cases, there was an agreement between the users of what label belonged to the tweet. This number is in the middle of the interval mentioned above, and therefore does not allow for us to draw conclusions on the validity of the experiment.

Figure 4.4: Histogram of human performance, showing the number of participants with a certain F1-score.

Table 4.13: Average F1-score of 110 survey participants vs. F1-score on LSTM network classi cations on the same dataset. Standard deviation of the neural network performance is based on 30 di erent runs.

	Survey	Network
F1-score	0.701	0.775
Standard Deviation	0.149	0.011

The results of this experiment showed that the network performed slightly better than the human participants, however the network performance value was within one half standard deviation. Assuming a normal distribution of human performance, we estimate that there is a 69 % chance that the network performs better than humans at this particular task. While one might expect humans to excel in sarcasm detection, we will not speculate too much on why the results showed the opposite. We would like to point out however, that a neural network has access to lots of similar data from its training part, while humans were not shown any data before the survey. Qualities such as sarcasm frequency or content of certain words might thus be features that the network is better accustomed to.

4.4 Quality of the datasets

The datasets that we used in this project were acquired from Twitter by two authors: The Sarcasm Detector and Ptá£ek et al. (2014), where the sarcastic tweets

were collected by using the sarcasm hashtag as, stated by Ptá£ek et al. (2014), an indicator for sarcastic tweets.

We believe that the said method for collecting sarcastic tweets was awed, since the presence of a sarcasm hashtag in a sentence does not necessarily indicate a sarcastic tweet. A concrete example of such a case is when it is used as a part of a sentence that is not otherwise sarcastic, see tweet id 1 in table 4.14. Additionally, since the datasets were collected from Twitter it meant that it was up to the Twitter users to correctly label their tweets in the case of a sarcastic tweet. This is an issue since sarcasm is a di cult concept to grasp for some people, which means that some tweets in the corpus labeled as sarcastic were non-sarcastic, see tweet id 5 in 4.14.

Another problem with the collection method is the strong correlation between the sarcasm hashtag and use of other hashtags among twitter users, as presented in section 4.2.3. This connection clearly in uence the outcome, as seen from the results in section 4.1. A sarcasm hashtag as indicator of sarcastic content might therefore not be the best choice.

Other aws that we noticed when examining the data were cases where tweets were not of English origin, bot-generated tweets or when the context for the sarcasm hashtag lied in another hashtag, see tweet id 2-4 in table 4.14.

The observations made in this section were lacking in both Ptá£ek et al. (2014), the creator of two of the datasets, and Poria et al. (2016) who used the said datasets in their work.

Table 4.14: Example of awed sample tweets from our datasets

Id	Sample tweet	Flaw
1	Jonathan is my name. #Sarcasm is my game.	#Sarcasm tag part of sentence
2	Por que todos los partidos del mundial son en Brasil? #sarcasm	Other language
3	#AcakFilm [rnaygn tge oruy ngu] (1940) Edith Conrad, p:gambling c:USA poin: 19	Bot-generated tweet
4	5:14 am and im still awake. #wonderful #sarcasm	Context in adjacent hashtag
5	I hate when I'm putting the nishing touches on something and my photoshop freezes and the crashes. #sarcasm	Not sarcastic

In addition to points made above, we noted that the tweets collected from the Twitter database do not comprise the entirety of the datasets D1 and D2 gathered by Ptá£ek et al. (2014), as stated in section 3.1. This means that we have only been able to train on approximately 75% of the data available to the previous researchers in the eld.

These points highlight that the data used in the twitter sarcasm detection eld is imperfect, and future works might want to look into alternate ways of data collection.

4.5 Obstacles

This section is dedicated to obstacles that were encountered during the course of the project.

4.5.1 Previous experience

Since none of our group members had any previous knowledge in the eld of machine learning the extensive period that would be dedicated to studying the eld was more complex than it should have been if prior knowledge had been possessed. Throughout the implementation process new concepts were introduced and pieces of necessary information were overlooked.

4.5.2 Finding quality data

One of our biggest obstacles, which we assume everyone pursuing this eld has, was to nd a solid quantity of quality data. We eventually came to a point where we had to settle with data that were used by previous academic papers such as Ptá£ek et al. (2014) and Poria et al. (2016). Our ambitions were presumably too grand in a way where we wanted to nd a dataset manually labeled.

4.6 Impact on society

In today's society, sarcasm presents problems not only due to verbal, but especially in written communication. If the research goes far enough to evolve NLP to a point where identifying sarcasm in real time text conversation or spoken language, there could be an aid for e.g. autistic people, who often have di culties classify sarcasm (Persicke et al., 2013). Ranick et al. (2013) writes that autistic kids more often are victims of bullying, since they often have trouble understanding non-literal language. This of course is nothing that any child, or adult for that matter, should have to withstand. Therefore sarcasm detection could help people who su ers from a mental condition such as autism.

As mentioned in the previous paragraph, detecting sarcasm surely would have positive outcomes. In today's eld of NLP, many tasks requires the understanding of the contextual sentiment given in any arbitrary text. As we previously talked about in chapter 1, we mentioned that Poria et al. (2016) claims that sarcasm is key for sentiment analysis for the fact that a sarcastic sentence has the ability to overturn polarity of the underlying sentiment.

5
Conclusion

From the results of the previous section we conclude that neural networks perform reasonably well in the task of predicting sarcastic content in tweets. They outperform a basic bag-of-words model in most cases and is on par with human ability. The proposed models are however inferior to current state of the art. One of the main aws with all our proposed models seem to be insu cient use of sequential information, as performance was only decreased with 4 % when words were scrambled.

Another weakness is that the network seems to struggle when evaluating across datasets, whose ratio between the sarcastic and non sarcastic partition di er. When training on D3, the performance is clearly worse when evaluating on D2 than D1 for all of the networks. An interesting result is that the bag-of-words model signi cantly performed better than all of the networks for the case when training was done on D2 and evaluating on D3. It seems as the e ect of going from a balanced to an unbalanced dataset a ects the neural networks more.

The performance decreased when tweets are cleaned of tags. It thus seems like there is a correlation between use of the sarcasm hashtag and other tags, which the networks picks up on. When cleaned of tweets, the networks did not perform better than a bag-of-words model. The agreement of 86 % between the classi cations of the network and bag-of-words model is a minor indicator that they might have some functionality in common.

The results presented in tables 4.2 - 4.4 varied very little with choice of network, as discussed in section 4.1. One and the same network did however achieve di erent scores on the di erent datasets. When training and evaluating the RNN with LSTM-cells on the dataset preprocessed by the strict means, explained in section 3.1.1, the score drops signi cantly. Additionally, preprocessing the data with tags yielded an increase in score by varying degree between the networks. These observations would imply that di erent basic deep learning archetypes are able to learn to classify to the same degree of accuracy, and that the achievable results are very much bound to the data itself.

Furthermore, our discussion of the quality of the datasets used in the report suggests that there is noise in the form of mislabeling, bot generated tweets and a strong correlation between sarcastic labels and tags content. Conclusively, future work which aim to identify sarcasm in text should primarily focus on gathering data better suited for the task.

References

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

Alex Graves. Supervised Sequence Labelling with Recurrent Neural Networks. *Image Rochester NY*, page 124, 2008. ISSN 01406736. doi: 10.1007/978-3-642-24797-2. URL https://arxiv.org/pdf/1308.0850.pdf .

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555 .

Tomá Ptá£ek, Ivan Habernal, and Jun Hong. Sarcasm detection on czech and english twitter. 2014. URL http://pure.qub.ac.uk/portal/files/17977967/Coling2014.pdf .

Aditya Joshi, Vinita Sharma, and Pushpak Bhattacharyya. Harnessing context incongruity for sarcasm detection. In *ACL (2)*, pages 757 762, 2015.

S. Poria, E. Cambria, D. Hazarika, and P. Vij. A Deeper Look into Sarcastic Tweets Using Deep Convolutional Neural Networks. *ArXiv e-prints*, October 2016. URL https://arxiv.org/pdf/1610.08815.pdf .

History World. History of language, 2017. URL http://www.historyworld.net/wrldhis/PlainTextHistories.asp?historyid=ab13 .

Cambridge. Meaning of sarcasm in the english dictionary, 2017. URL http://dictionary.cambridge.org/dictionary/english/sarcasm .

Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51 89, jan 2005. ISSN 00664200. doi: 10.1002/aris.1440370103. URL http://doi.wiley.com/10.1002/aris.1440370103http://arxiv.org/abs/0812.0143{%}5Cnhttp://doi.wiley.com/10.1002/aris.1440370103 .

US Secret Service. Computer based annual social media analytics subscription. 2014. URL https://www.fbo.gov/?s=opportunity&mode=form&id=8aaf9a50dd4558899b0df22abc31d30e&tab=core&_cview=0

Li Deng and Dong Yu. *Deep Learning Methods and Applications*. now publishers Inc, 2014.

Qv Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. International Conference on Machine Learning - ICML 2014, 32:1188 1196, 2014. ISSN 10495258. doi: 10.1145/2740908.2742760. URL http://arxiv.org/abs/1405.4053 .

Aditya Joshi, Pushpak Bhattacharyya, Mark James Carman, Jaya Saraswati, and Rajita Shukla. How do cultural di erences impact the quality of sarcasm annotation?: A case study of indian annotators and american text. Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities, W16-2111, 2016a. URL https://aclweb.org/anthology/W/W16/W16-2111.pdf.

Aditya Joshi, Pushpak Bhattacharyya, and Mark James Carman. Automatic sarcasm detection: A survey. CoRR, abs/1602.03426, 2016b. URL http://arxiv.org/abs/1602.03426 .

A Ghosh and T Veale. Fracking sarcasm using neural network. Proceedings of NAACL-HLT , 2016. URL http://anthology.aclweb.org/W/W16/W16-0425.pdf .

Ellen Rilo , Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. Sarcasm as Contrast between a Positive Sentiment and Negative Situation. 2013. URL https://www.cs.utah.edu/{~}riloff/pdfs/official-emnlp13-sarcasm.pdf .

Dmitry Davidov, Oren Tsur, and Ari Rappoport. Enhanced Sentiment Learning Using Twitter Hashtags and Smileys. Proceedings of the 23rd International Conference on Computational Linguistics: Posters, (August):241 249, 2010. doi: 10.1.1.185.3112. URL http://dl.acm.org/citation.cfm?id=1944566.1944594 .

Will Knight. The dark secret at the heart of ai, 2017. URL https://www.technologyreview.com/s/604087/the-dark-secret-at-the-heart-of-ai/ .

Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. Identifying sarcasm in Twitter: a closer look. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2, (2010):581 586, 2011. doi: 10.1.1.207.5253. URL http://www.aclweb.org/anthology/P/P11/P11-2102.pdf .

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4):115 133, 1943. ISSN 00074985. doi: 10.1007/BF02478259.

F Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386 408, 1958. ISSN 0033-295X. doi: 10.1037/h0042519.

David E Rumelhart, Geo rey E Hinton, and R J Williams. Learining Internal Representations by Error Propagation, 1986. ISSN 1-55860-013-2.

J.S. Bridle. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. Neurocomputing: Algorithms, Architectures and Applications, (C):227–236, 1990. doi: 10.1007/978-3-642-76153-9.

David Kriesel. A Brief Introduction to Neural Networks, 2007. ISSN 14320711. URL http://linkinghub.elsevier.com/retrieve/pii/0893608094900515 .

George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pages 8609–8613. IEEE, may 2013. ISBN 978-1-4799-0356-6. doi: 10.1109/ICASSP.2013.6639346. URL http://ieeexplore.ieee.org/document/6639346/ .

Yann LeCun, L??on Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2323, 1998. ISSN 00189219. doi: 10.1109/5.726791.

Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks, 2015. URL http://karpathy.github.io/2015/05/21/rnn-effectiveness/ .

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. A Field Guide to Dynamical Recurrent Networks, pages 237–243, 2001. ISSN 1098-6596. doi: 10.1109/9780470544037.ch14. URL http://www.bioinf.jku.at/publications/older/ch7.pdf .

Sepp Hochreiter and Jurgen Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1–32, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. CoRR, abs/1409.1259, 2014. URL http://arxiv.org/abs/1409.1259 .

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. CoRR, abs/1502.02367, 2015. URL http://arxiv.org/abs/1502.02367 .

Kevin Roebuck. Lightning Source, 2011. ISBN 1743046316, 9781743046319. URL https://www.iho.int/mtg_docs/com_wg/TSMAD/TSMAD22/TSMAD22_DIPWG3-11.7A_S-101_Data_Quality_FINAL.pdf.

Charles Elkan. The foundations of cost-sensitive learning. Department of Computer Science and Engineering 0114, University of California, San Diego, 2001. URL https://pdfs.semanticscholar.org/0fba/3766c7d613da8f35a2872f728c0c9e081092.pdf.

Sebastian Ruder. An overview of gradient descent optimization algorithms. Web Page, pages 1–12, 2016. URL https://arxiv.org/pdf/1609.04747.pdf http://arxiv.org/abs/1609.04747.

David White and Panos Ligomenides. GANNet: A Genetic Algorithm for Optimizing Topology and Weights in Neural Network Design. In New Trends in Neural Computation, pages 322–327. Springer, Berlin, Heidelberg, 1993. ISBN 3540567984. doi: 10.1007/3-540-56798-4_167. URL http://link.springer.com/10.1007/3-540-56798-4_167.

Eric W. Weisstein. Gradient. URL http://mathworld.wolfram.com/Gradient.html.

Léon Bottou. Online learning and stochastic approximations. AT&T Labs Research, Red Bank, NJ 07701, 1998. URL http://leon.bottou.org/publications/pdf/online-1998.pdf.

Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. arXiv, pages 1–14, 2014. ISSN 10495258. URL http://arxiv.org/abs/1406.2572.

Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations, pages 1–13, 2014. ISSN 09252312. doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. URL http://arxiv.org/abs/1412.6980.

M. Nielsen. Neural network and deep learning, 2015. URL http://neuralnetworksanddeeplearning.com/chap2.html.

Yann Le Cun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. 1:21–28, 1988.

R. Hecht-Nielsen. Theory of the backpropagation neural network. pages 593–605 vol.1, 1989. doi: 10.1109/IJCNN.1989.118638.

Christopher M Bishop. Pattern Recognition and Machine Learning, volume 4. 2006. ISBN 9780387310732. doi: 10.1117/1.2819119. URL http://www.library.wisc.edu/selectedtocs/bg0137.pdf.

Nitish Srivastava, Geo rey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from over tting. Journal of Machine Learning Research, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Tom Fawcett. An introduction to ROC analysis. Pattern Recognition Letters, 27 (8):861–874, 2006. ISSN 01678655. doi: 10.1016/j.patrec.2005.10.010.

Charles Parker. On measuring the performance of binary classi ers, 2013. ISSN 02191377.

D.M.W. Powers. Evaluation: From Precision, Recall and F-Measure To Roc, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011. ISSN 2229-3981. doi: 10.1.1.214.9232. URL http://www.bioinfopublication.org/files/articles/2{_}1{_}1{_}JMLT.pdf.

Y Bengio, R Ducharme, P Vincent, and C Jauvin. A neural probabilistic language model. *Journal of machine learning*, 2003. URL http://www.jmlr.org/papers/v3/bengio03a.html.

Tomas Mikolov, Kai Chen, Greg Corrado, and Je□rey Dean. E□cient Estimation of Word Representations in Vector Space. URL https://arxiv.org/pdf/1301.3781.pdf.

Je□rey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. URL https://nlp.stanford.edu/pubs/glove.pdf.

Angela Persicke, Jonathan Tarbox, Jennifer Ranick, and Megan St. Clair. Teaching children with autism to detect and respond to sarcasm. *Research in Autism Spectrum Disorders*, 7(1):193–198, 2013. ISSN 17509467. doi: 10.1016/j.rasd.2012.08.005. URL http://www.sciencedirect.com/science/article/pii/S1750946712000980.

Jennifer Ranick, Angela Persicke, Jonathan Tarbox, and Jake A. Kornack. Teaching children with autism to detect and respond to deceptive statements. *Research in Autism Spectrum Disorders*, 7(4):503–508, 2013. ISSN 17509467. doi: 10.1016/j.rasd.2012.12.001. URL http://www.sciencedirect.com/science/article/pii/S1750946712001481.