WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Bachelor Thesis

Evaluating Document Level Sentiment Analysis Models: CNN vs RNN

Elias Haueis

Date of Birth: 16.03.1998
Student ID: 11704657

Subject Area: Information Business

Studienkennzahl: UJ 033 561

Supervisor: ao.Univ.Prof. Dr. Johann Mitlöhner

Date of Submission: 31.10.2022

Department of Information Systems and Operations, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

DEPARTMENT FÜR INFORMATIONSVERARBEITUNG UND PROZESSMANAGEMENT DEPARTMENT OF INFORMATION SYSTEMS AND OPERATIONS

Contents

List of Figures

List of Tables

Abstract

In order to find the opinions of the authors of texts on a large scale, neural networks are one possible approach in order to achieve state of the art classification performance. When conducting document level sentiment analysis with supervised learning, many different viable possibilities for model architecture exist. This thesis shows how to implement different model-architectures, namely convolutional neural networks, recurrent neural networks (LSTM), as well as a hybrid-solution. The different models are trained on the same dataset, using similar parameters in order to compare the performance.

1 Introduction

1.1 Motivation

With the increase of user generated data on websites like social media platforms or e-commerce websites, there is an increased demand for tools which are able to automatically extract information and insights from said data. Recent advances in Deep Learning (DL) had a significant impact on the field of natural language processing in general and on sentiment analysis in particular. These advances combined with the availability of increasingly powerful graphical processing units (GPU), which enable deep learning models to train very fast, are leading to deep learning models becoming one of the most performant solutions for sentiment analysis tasks. The use cases range from businesses, which want to better understand the feelings and opinions of their customers to individuals and even governments, which have great interest in the public opinions about policies and elections [8]. On a more abstract level, models for sentiment analysis can be used for reputation management, market research, competitor analysis or product analysis [24].

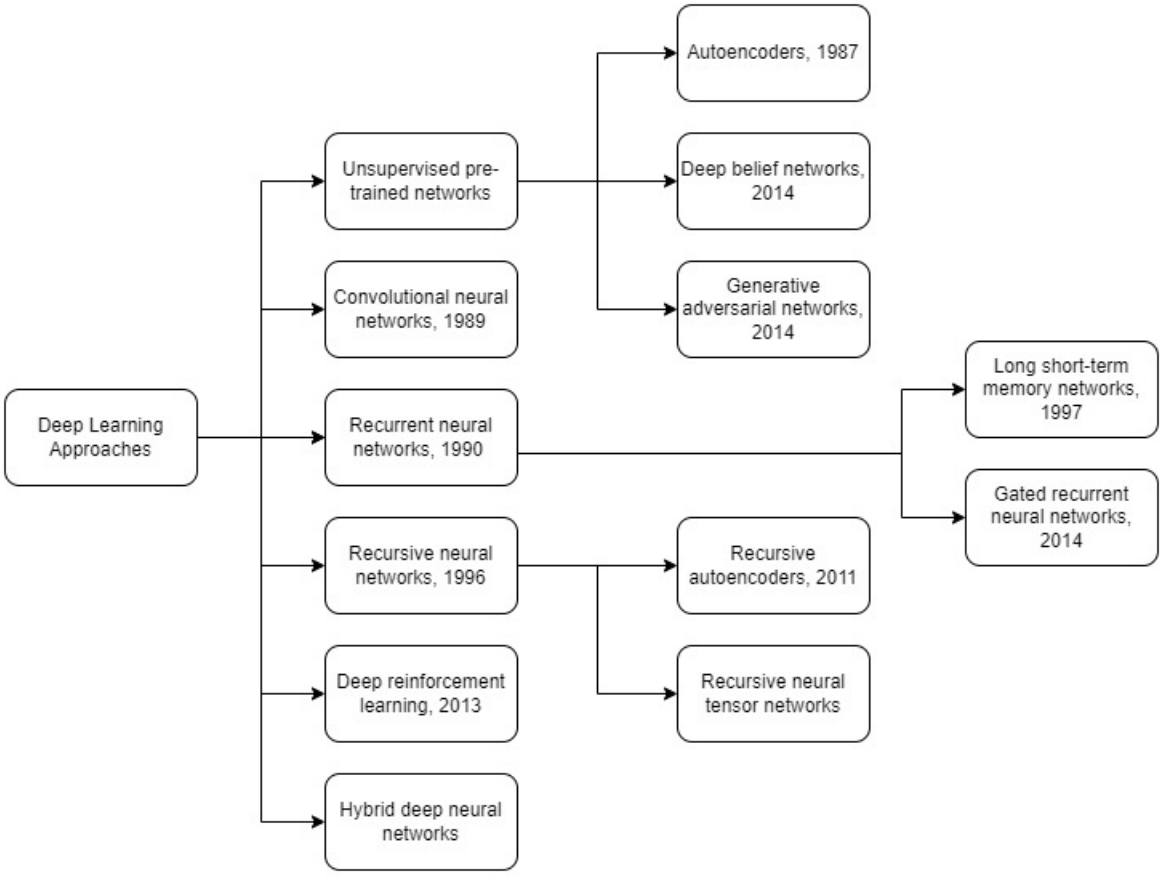

Figure 1: Deep Learning Approaches for Sentiment Analysis, Habimana et al.

[8]

1.2 Research Question

The overall goal of this thesis is to design and implement three different deep learning models for document level sentiment analysis, evaluate the performance of said models and compare and contrast the different implementations. The approach is formalized in the following research question:

How do different deep learning models, namely convolutional neural networks and recurrent neural networks perform at sentiment analysis?

1.3 Approach

Overall, there are many different deep learning approaches which are usable for sentiment analysis. Figure 1 gives an overview of the viable approaches. Some of the approaches vary significantly in their underlying functionality. For supervised, document level sentiment analysis there are many viable options for the architecture of the model. In this work, the focus lies on convolutional neural networks, recurrent neural networks as well as hybrid solutions, which have all shown to provide state of the art performance in the field of sentiment analysis. To answer the research questions mentioned in the previous section, we are going to implement one of each mentioned models, train them on the same data set and evaluate their performance. This work, including all of the code can be found at https://github.com/ehaue1s/ba-thesis.

1.4 Structure

After a brief description of the method (chapter 2), the theoretical background of deep learning and natural language processing will be laid in detail (chapter 3), in order to cover the necessary foundations of this work. Chapter 4 covers related work, particularly similar projects and approaches where we derive useful procedures, techniques and insights to improve the implementation. In chapter 5 the implementations of the different models are described in detail. In the following chapter 6 the models created are validated, tested and ranked based on different metrics. In the final chapter, the results and limitations will be discussed.

2 Method

This work follows the classical episodic design science research method [2]. This method is usually structured as follows:

1. Search
2. Design
3. Ex ante evaluation
4. Construction of the artifact
5. Ex post evaluation

Based on the theoretical background (chapter 3) and related work (chapter 4) different approaches are collected and evaluated. In chapter 5, the implementations of the artifacts are described based on the underlying groundwork done in the previous chapters. After this step, the artifacts are evaluated in chapter 6 and limitations will be discussed in chapter 7.

3 Theoretical Background

3.1 Artificial Intelligence

Up to this point there is no general and widely accepted definition of artificial intelligence (AI). The field consists in fact of a mixture of multiple research fields with their respective goals, methods and applications [23]. Generally speaking, artificial intelligence describes the automatization of intellectual tasks usually performed by humans [3]. Over the years, the field of AI went from relying on hard-coded knowledge about the world in combination with inference rules to systems, which have the ability to acquire knowledge by extracting patterns from raw [7]. One of these "traditional" programming approaches in AI is called symbolic artificial intelligence, where human knowledge of a specific task is hard coded as a specific rule set into the program. This approach does not include any automated learning of the program itself, simply every possible scenario of the task to be automated would be covered within the program [3]. However, systems which rely on hard-coded knowledge were not very successful over the history of AI and led to the development of systems with the capability to generate knowledge from data by themselves. This capability is generally called machine learning (ML) [7].

3.2 Machine Learning

Machine learning (ML) in general centers around the learning aspect of artificial intelligence. The goal is to develop algorithms that represent a set of data very accurately. Whereas in classical programming, the computer is supplied with a dataset and an algorithm to produce some kind of output, in ML the computer generates an algorithm that describes the relationship between a given dataset and its associated output. Machine learning approaches are very suitable for tasks which profit from high-level pattern recognition, e.g. image classification, where traditional programming approaches usually struggle in terms of performance and accuracy [3]. There exist a variety of ML approaches, which are used today in various fields. One of the more simplistic ML algorithms is called logistic regression, which is used in medicine

to give recommendations regarding cesarean delivery. Another everyday use case of ML is spam detection in the communication via e-mail, where an algorithm called naive Bayes is used to differentiate spam from legitimate messages [7].

3.2.1 Machine Learning Approaches

Machine learning can be further classified into 4 different approaches: supervised learning, semi-supervised learning, unsupervised learning as well as reinforcement learning. Since this thesis follows a supervised learning approach we are going to focus on this approach.

Supervised learning Besides the other approaches mentioned above, supervised learning is the most common form of machine learning [15]. Concerning the training data, the data set has to consist of examples for the input, as well as previously labeled answers or target values for the output. The input-output pairs are then used to calibrate the parameters of the ML model during the training process. If the model has been successfully trained, the model is able to predict the target variable based on new or unseen data points of the input feature [11].

Comparison to other ML approaches Compared to the supervised learning approach, unsupervised learning is intended to detect patterns in a data set without pre-existing labels. Unsupervised learning aims to detect patterns or structural information, like clustering or dimensionality reduction. Another ML approach would be reinforcement learning, which is an approach based on trial and error. One can specify a goal and a list of allowable actions including their environmental constraints. Based on these conditions and a description of the current state of the system, the model tries different solution in order to achieve the previously defined goal [11].

3.2.2 Generalization

One of the main goals of a machine learning algorithm is to perform well on new, unseen input data. This capacity is referred to as generalization. In order to measure how well a model performs on previously unseen data, a test data set is introduced to compute indicators for the model performance. During the training process a training error is computed. The model is trained by minimizing the training error. The other important indicator is the test error or generalization error. The test error is the expected value of error for new inputs. Machine learning models in general are intended to

have minimal test error. In order to ensure good performance, the training error, as well as the gap between training and test error have to be minimal. If the model is not able to obtain a sufficiently low training error, the model is underfitting. On the other hand, if the gap between training and test error is too large, the model is overfitting [7].

3.2.3 Hyperparameters

Hyperparameters can be defined as settings for controlling the behavior of a machine learning algorithm. Hyperparameters in general are not adopted by the learning algorithm. These parameters are usually difficult to optimize or difficult to learn on the training data set [7].

3.2.4 Validation

A validation set is introduced to the model in order to ensure that no overfitting is taking place. The validation set is usually constructed from the training set, but it must not be observed by the learning algorithm. The training data is therefore split into a training set as well as a validation set, where 80 percent of the data is usually used for training and the rest for validating the model. While the training data sets the parameters of the model during the training process, the validation set is used to set the hyperparameters accordingly. This is also part of the training process. Any modification of the model to reduce its generalization error is also called regularization [7].

3.2.5 Testing

After training is complete, the model can be tested and the generalization error can be estimated through the test set, which is independent from the training data [7].

3.2.6 Data representation

Data representation is one of the key factors of ML which impacts performance. The data used for training a ML model usually consists of different types of information, which are called features. Depending on how the features are represented in the training data, the performance of the ML model will vary. One example for different representations of the same data would be polar coordinates and Cartesian coordinates. In classical Machine Learning as well as rule-based AI systems, features are in general hand-designed.

The problem with hand-picked features is that it is very difficult for humans to decide which features to pick and what representation to use [7].

3.3 Representation Learning

In order to solve the problems hand-picked features are causing, ML models were further developed to discover the mappings of the data, as well as the representation of the data itself. This approach is called representation learning and can be classified as a sub-field of machine learning. In general, representation learning is faster than designing the features by hand. One example of a representation learning algorithm is the autoencoder. This algorithm uses an encoder function to convert the input in various representations and a decoder function to convert them back into their original format [7].

3.4 Artificial Neural Networks

Artificial neural networks (ANN) are machine learning algorithms, which are inspired by biological neural networks like our human brain [3]. Concerning their structure, ANNs consist of mathematical representations of interconnected processing units. These units are called artificial neurons [11]. Artificial neurons are analogue to the cell bodies of a biological neural network, while the connections between them represent the axons and dendrites [3]. The connections between the artificial neurons transmit signals of varying strength based on a weight which is continuously adjusted during the learning process. Determined by an activation function, each subsequent neuron is only activated, if a certain threshold is exceeded. Within the network, the artificial neurons are organized in different layers. Besides input and output layers, an ANN consists of zero or more hidden layers [11].

3.4.1 Activation Function

Activation functions are essential for the performance of an ANN. As already mentioned, the layers of the network consist of interconnected artificial neurons, where each neuron has an activation function associated with itself. Generally speaking, an activation function is used to transform an input signal into an output signal, which then acts as an input to the next node or layer. The performance of an ANN is heavily influenced based on what activation function is used. Overall, there are many different activation functions to choose from, but mostly non-linear functions are used. This is due to the fact that there is a need for non-linearity in neural networks. A linear

function might be easy to solve and computationally inexpensive, however, it lacks the ability to learn complex mappings from a data set. To extract more complicated information and achieve non-linear mappings from inputs to outputs, we need to use non-linear activation functions. Another important requirement for activation functions in ANN's is that they need to be differentiable. This is needed to implement a backpropagation (chapter 3.5.2) strategy to compute errors/losses, as well as to optimize weights to reduce errors using one of several optimization techniques [22].

Here are some of the activation functions explained in detail. Note that only activation functions in relevance to this project are mentioned.

Rectified linear unit (ReLU) The ReLU function is one of the most common activation functions used in neural networks.

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

The associated neuron is only activated, if the output of the linear transformation is greater than 0. Due to the fact that not all neurons are activated at the same time, the ReLU function in general is more efficient than most activation functions. This also has an impact on backpropagation (chapter x), where if the function's output is equal to zero, the weights of the model are not updated for backwards connected neurons [22].

Sigmoidal The sigmoid activation function is another nonlinear activation function. The function transforms the values of the input to the range of 0 to 1.

$$f(x) = \frac{1}{e^{-x}}$$

The function is continuously differentiable, which enables backpropagation. The sigmoidal activation function is used for binary classification problems [22].

Hyperbolic tangent function (Tanh) Similar to the sigmoidal activation function, the hyperbolic tangent function transforms values into the range of -1 to 1.

$$f(x) = 2sigmoid(2x) - 1$$

The function is also continuously differentiable, but compared to the sigmoidal activation function steeper[22].

Softmax The softmax activation function consists of a combination of multiple sigmoid functions and can be described as follows:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e_{z^k}}$$

The function returns a probability for every datapoint of all the individual classes. It can therefore be used for multiclass classification problems [22].

3.5 Deep Learning

3.5.1 Overview

Deep learning is a concept within machine learning which is based on artificial neural networks [11].

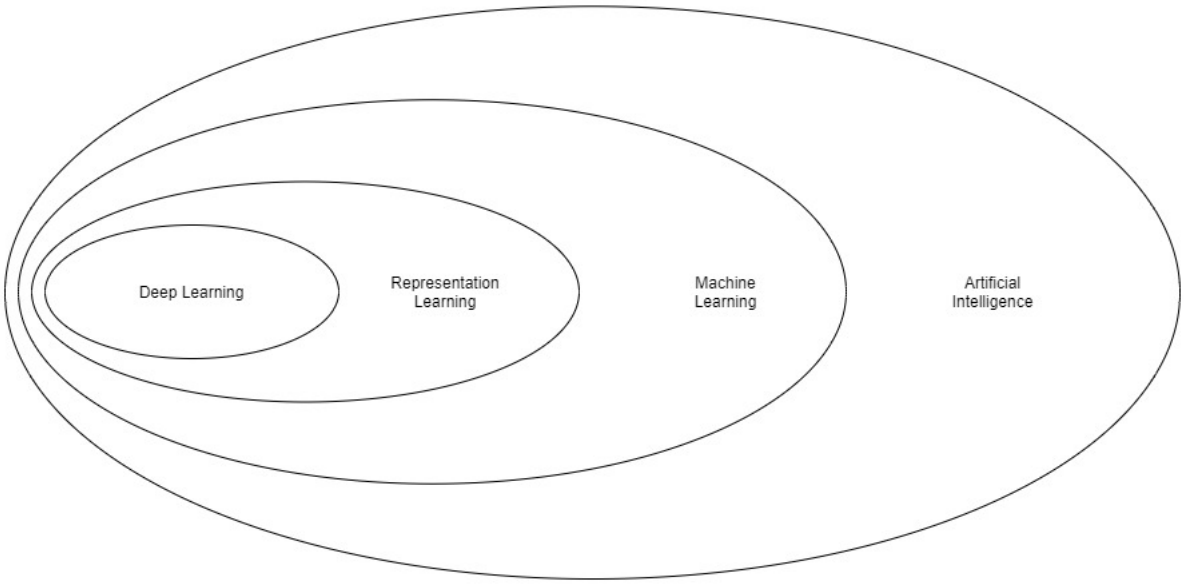

Figure 2: Artificial Intelligence Overview, Goodfellow [7]

In Figure 2 we can see an overview of AI and its various sub-disciplines as classified by Goodfellow. He classifies DL as a subcategory of representation learning. According to Goodfellow, the quintessential example of a deep learning model is the multilayer perceptron or feedforward deep network. Although the terms to describe these kinds of models may vary, they have some distinct features, which differentiate them from generic machine learning models. In comparison to standard artificial neural networks, deep neural networks consist of more than one hidden layer [11]. However, there is no consensus on how much depth is required for a model to qualify as "deep" [7].

Because of this advanced complexity, they are typically organized in deeply nested network architectures. These differences enable improvements in contrast to standard ANN's: It is possible to implement layers made of advanced neurons, which are able to use advanced operations like convolutions or multiple activations per neuron, while the neurons in conventional ANNs rely on a simple activation function [11]. Another distinction can be made in terms of data representation, where deep neural networks are able to extract high level, abstract features from raw data due to their multi layer architecture [7].

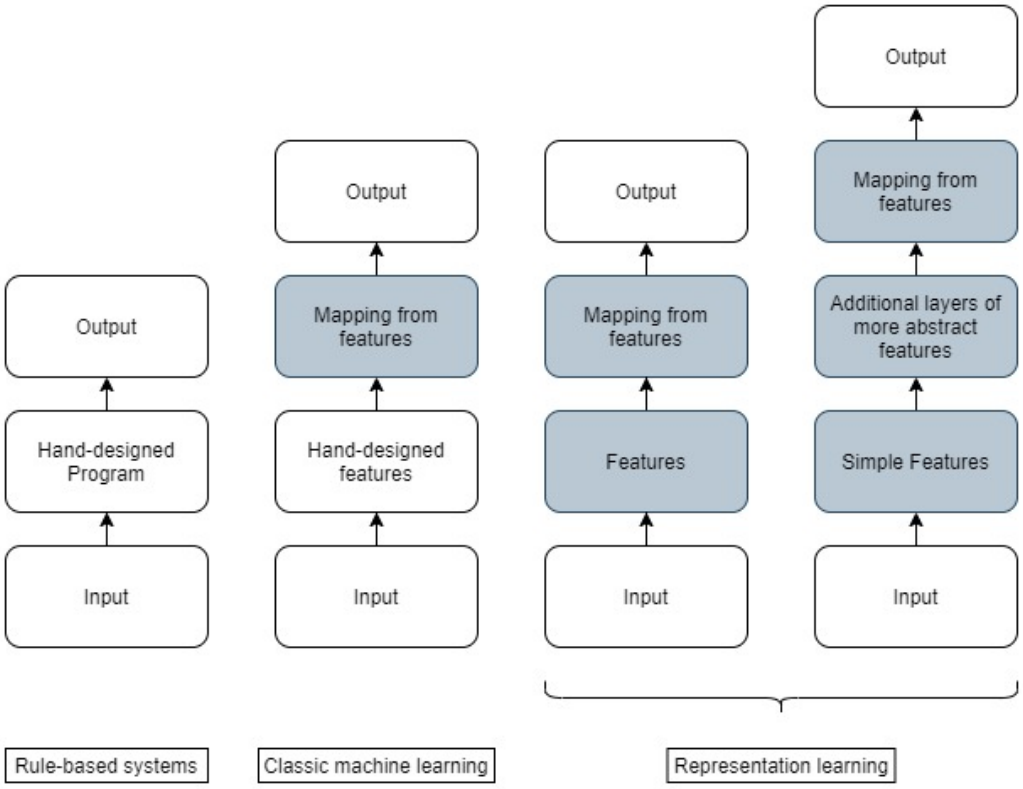

Figure 3: Differences between AI Branches, Goodfellow [7]

Generally speaking, the goal of a feed-forward deep learning model is to approximate a function f*.

$$y = f(x, \Theta)$$

y – output,
x – input,
Θ – parameter.

During the training process, the network learns values for the parameters Θ in order to give the best approximation of the function, resulting in f*.

$$y = f(x, \Theta) \rightarrow y = f^*(x)$$

These networks are generally called feedforward networks, because the information flows from the input x to the output y in order to define the function f* through the immediate computations. In basic ANNs, there are no feedback loops from the output of the model back into the model itself. If such connections are implemented, the model is referred to as a recurrent neural network (RNN), which will be explained in detail in chapter3.5.4 [7].

Depth Usually, deep neural networks are composed of many different functions. Concerning the depth of the model, Goodfellow describes the following example: Given three functions f1, f2 and f3, a chain structure can be formed, which is often used in neural networks.

$$y = f3(f2(f(x))$$

The length of the chain equals the depth of the model. In this case the model would have a depth of three [7].

Layers Deep Learning models consist of a series of different layers. Looking at the example from the previous chapter, the function f1 would be equivalent to the input layer, which can also be referred to as a visible layer. On the other hand, the final layer of the network is called the output layer(f3). Besides the input and output layer, the behavior of the other layers is not directly specified by the training data. The learning algorithm itself decides how to use the layers in order to achieve the best approximation of the function f*. The layers between input and output layer are therefore called hidden layers. In the example from above, the function f2 would be the single hidden layer [7].

3.5.2 Optimization

Optimization is another important requirement for the learning process of deep learning algorithms. In the context of deep learning, optimization stands for maximizing or minimizing a function f(x) by altering x [7].

Gradient based optimization In deep learning, the function which is minimized during the optimization process is usually called cost function or loss function. Minimizing a function f(x) is done through calculating its derivative with respect to its weights. The points where the derivative of said function is equal to zero are called critical points. The absolute lowest point of a function is generally referred to as the global minimum. However, not

every point where $\frac{\partial f}{\partial x} = 0$ is a global minimum, because the point can also be a local minimum, saddle point or local/global maximum. A function can also have multiple global minima. Overall, there are many tasks in the field of DL, which require the gradient either during the training process or to analyze the model [7]. One of the most common optimization algorithms in the area of deep learning is called gradient descent. The algorithm updates the parameters of a function in the opposite direction of the gradient of the objective function. The size of the steps taken to reach the local minima of the function is usually referred to as its learning rate. Many variants of this algorithm exist, such as batch gradient descent, stochastic gradient descent or mini-batch gradient descent. The variants mainly differ in how much data is used to compute the gradient. However, these algorithms suffer from certain drawbacks. It is for example very difficult to choose a proper learning rate: If the learning rate is too small it leads to slow convergence, if it is too high it can cause the loss function to fluctuate around the minimum or even to converge. Another problem might be, that the algorithm gets trapped on one of the numerous local minima or saddle points, which is sub-optimal in terms of performance. To circumvent said problems, numerous gradient descent optimization algorithms have been introduced, like Momentum, Nesterov accelerated gradient, Adagrad, Adadelta and Adam. Although some of them are very similar, every one of these algorithms has its advantages and disadvantages [21].

Adam The Adam algorithm is an algorithm for first order gradient based optimization and was introduced in 2015. The name Adam is derived from adaptive moment estimation. The algorithm is computationally efficient and has little memory requirements. Furthermore, it is very suitable for problems which involve a lot of data or parameters[13].

Backpropagation The flow of information through a neural network from the input layer through the several hidden layers into the output layer is called forward propagation. During the training process, the forward propagation eventually produces scalar cost. The back-propagation algorithm, which is another important component of some artificial neural networks, uses information about the cost to flow back into the network in order to compute the gradient of a function. The term back propagation itself refers to the method of computing the gradient [7]. This is accomplished by applying the chain rules of derivatives [15].

3.5.3 Convolutional Neural Networks

In most cases, convolutional neural networks are applied to tasks like computer vision and speech recognition. Generally speaking, they are suitable for tasks centered around data sets with spatial relationships where columns and rows are not interchangeable [11]. In other words, CNNs are designed to process data, which consists of multiple arrays. For computer vision, 2-dimensional arrays are used, while sequence based data, like language, usually comes in the form of 1-dimensional arrays [15].

Convolutional neural networks are based around 4 key ideas: local connections, shared weights, pooling and the usage of many layers. In general, they consist of convolutional layers and pooling layers. Convolutional layers are responsible for detecting local conjunctions of features from the previous layer. Pooling layers on the other hand merge semantically similar features into one [15].

The convolutional layers are organized in feature maps. Each of the units inside of a feature map is connected to local patches in the feature map of the previous layer. The units are connected through so-called filter banks: A set of weights which is shared by all units in the filter bank. Each local feature map uses a different filter bank. This is due to the fact that in array based data structures, such as images, local groups of variables are highly correlated. The results of the weighted sum are then mapped through a nonlinear function, such as ReLU, in order to achieve nonlinear mappings [15].

3.5.4 Recurrent Neural Networks

Recurrent neural networks (RNN) are specialized for processing sequential data, e.g. language. One of the key ideas of RNNs is the sharing of parameters across different parts of the model [7]. The architecture further involves internal feedback loops, which enables the model to learn time dependencies by forming a memory [11]. This was first made possible through the introduction of backpropagation to the model [15].

Although RNNs were innovative for dealing with sequential inputs, simple RNN suffer from one problem: the back propagated gradients change at each time step the model. This leads to the gradients either blowing up or vanishing during the training process [15]. This has an impact on the efficiency of the algorithm: “The temporal evolution of the back propagated error exponentially depends on the size of the weights” [10]. This lead to

the development of special versions of RNNs which use hidden units with an explicit memory, like Long-short term memory [15].

Long-short term memory LSTM In 1997 Hochreiter introduced a solution for the error-backflow problem: Long Short-term memory(LSTM). LSTM uses an efficient, gradient-based algorithm which enables a constant error flow through the internal states of the model units. This is accomplished by the implementation of a LSTM-cell [10]. A LSTM-Layer consists of recurrently connected memory blocks. Every memory block is made of one or more recurrently connected memory cells. Each of these cells have special multiplicative gate units: Besides the standard input and output of the cell, there is an input gate, an output gate as well as a forget gate. The special gates perform functions on the cells memory, which are equivalent to the operations read, write and reset. This enables the network to store information over long periods of time, without causing vanishing or exploding gradients [26].

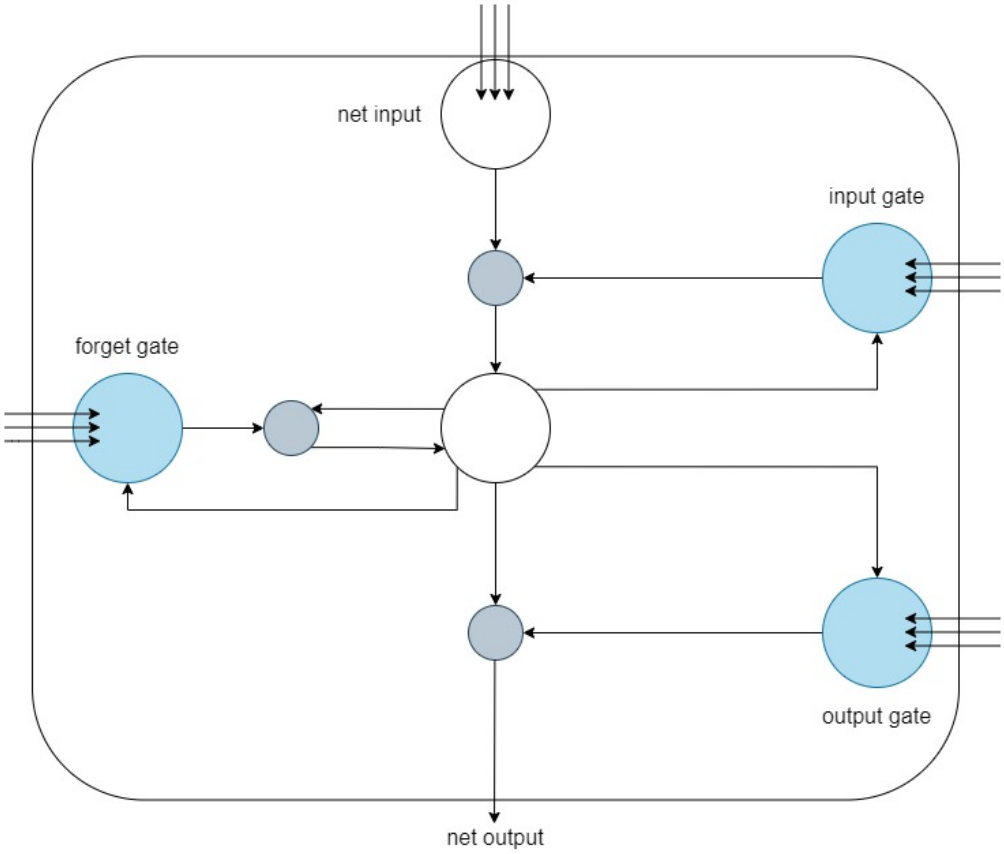

Figure 4: LSTM Cell, Wöllmer
[26]

3.6 Natural Language Processing

Natural language processing, also known as computational linguistics, can be classified as a subfield of computer science with the purpose of understanding, learning and producing human language content with computational techniques. Over the last 25 years the field became increasingly important and found usage in a lot of areas and products. Some examples for real-worlds applications would be machine translation, speech recognition and speech synthesis [9].

3.6.1 Sentiment Analysis

Sentiment analysis is closely related to computational linguistics, natural language processing and text mining. The term can be paraphrased as subjectiv-

ity analysis, opinion mining, or appraisal extraction. The objects of study in this field are opinions and subjectivity. These "subjective elements", after the original definition by Wiebe et al. consist of "linguistic expressions of private states in context" [25]. Said elements can be differentiated by their length. Analysis is possible for a single word, phrase, sentence or document. There are several tasks associated with sentiment analysis: sentiment detection, polarity classification, discovery of the opinion's target and feature extraction. Sentiment detection is about classifying a text as subjective or objective. This is usually done by examining a sentence's adjectives or adverbs. The goal of polarity classification is to classify the opinion of the author of a text into two opposing sentiment polarities (e.g. positive or negative). This can be done via a binary classification or via a multi-class classification, using a multi-point scale like for example 0-10 [17]. This thesis also falls under the term polarity classification. The discovery of the opinion's target on the other hand is very complicated and highly depends on the domain of the analysis. Lastly, feature extraction is about extracting components or attributes of a text, which are called features [17].

The general workflow of a sentiment analysis task is as follows: First, the input data has to be preprocessed using linguistic tools like stemming, tokenization, entity extraction, speech tagging and relation extraction. The main component of the sentiment analysis system is then used to annotate the preprocessed data with its sentiments. Finally, the output of the system can be presented using data visualization [6].

Sentiment Analysis can be further broken down into five categories: Document-level sentiment analysis, sentence-level sentiment analysis, aspect-based sentiment analysis, comparative sentiment analysis and sentiment lexicon acquisition. Document level sentiment analysis is the simplest form of sentiment analysis. This can be used on texts consisting of one opinion of the user on one main object. This is usually done by following a supervised or unsupervised ML approach. If a more fine grained view on the opinions of the author is needed, a document must be divided into its sentences and sentiment analysis has to be conducted on each entity separately. It is therefore assumed, that each sentence consists of one opinion. If these approaches are not detailed enough for our needs, the aspect-based sentiment analysis approach can be used. In general, aspect-based sentiment analysis is the research problem, which focuses on obtaining all sentiment expressions within a document and the aspects they refer to. The comparative sentiment analysis approach is about the identification of sentences, that contain comparative opinions and their extraction. The approach heavily depends on comparative

adjectives and adverbs, superlative adjectives and adverbs as well as additional phrases like for example “favor”, “exceed” or “outperform”. Lastly, the sentiment lexicon acquisition refers to all approaches to obtain a so-called sentiment lexicon: The sentiment lexicon is a crucial resource for many sentiment analysis algorithms, like for example unsupervised ML approaches [6].

3.6.2 Document Level Sentiment Analysis

As the title of this thesis suggests, this work is about the implementation of several document level sentiment analysis models. In general, there are two main approaches for sentiment analysis on the document level: supervised learning and unsupervised learning. The supervised approach is based on the assumption, that the document should be classified into a finite set of pre-defined classes with certain attributes. The simplest case would be the binary classification, e.g. positive or negative. This approach can be extended by adding a neutral class, depending on a discrete numeric scale or adding a sentiment analysis lexicon into the algorithm. Several machine learning algorithms are suitable to perform such a classification. In order to perform the supervised learning approach, enough training data for each class is indispensable. Unsupervised approaches on the other hand depend on determining the semantic orientation of phrases within the document. Within unsupervised learning, there are two main approaches: predefined part-of-speech patterns and sentiment lexica [6].

3.6.3 Sentiment Analysis using Deep Learning

In recent history, deep learning has been used more frequently in the field of sentiment analysis. There are a lot of approaches within deep learning, which are applicable in the field of sentiment analysis. Besides unsupervised pre-trained networks, recursive neural networks and deep reinforcement learning, convolutional neural networks and recurrent neural networks have shown promising results in sentiment analysis [8].

Word Embeddings In order to make language suitable for the training process of a neural network, we have to pre-process the text data. One of the most common representation techniques are word embeddings. Word embeddings serve as the first data processing layer within a deep learning network. The texts therefore have to be represented in the form of vectors, where words with similar meaning and context are represented by similar vectors [8].

CNN and RNN for NLP Convolutional neural networks and recurrent neural networks promising approaches in the field of natural language processing. They are widely explored for different types of tasks within NLP. CNN have been shown to be very good at extracting position-invariant features. RNN have an advantage to other DL approaches in terms of sequence modeling. They are therefore both very suitable for NLP [27].

4 Related Work

In 2017, Yin et al. conducted a comparative study about the usage of CNN and RNN in natural language processing. In detail, they compared the usage of CNN, LSTM and GRU (gated recurrent unit, a type of RNN similar to LSTM) in a representative sample of NLP tasks. They trained all their networks from scratch (no pre-trained word embeddings), did not use complex tricks like batch normalization and used different optimal hyperparameters for each model and task. Their main findings can be summarized as follows: Out of the 7 tasks the models were tested on, GRU performed best in 5 of them, while CNN was best in 2 categories. However, the results were overall very similar and deviations were minimal. They also found a high sensitivity to varying hyperparameters. Although all the tested models were very smooth in terms of learning rate, variations in the parameters hidden size and batch size caused performance to vary dramatically [27].

A relatively new approach was proposed by Ankita et al. in 2021. In their work they proposed an efficient CNN-LSTM model architecture specifically for sentiment detection. The hybrid model was used to detect the sentiments and emotions of twitter communication with regards to the #BlackLivesMatter movement. In their model, they combined CNN and LSTM into their architecture. Figure x shows their proposed architecture in detail. They compared the performance of the model to different other architecture types, including a pure CNN approach as well as a pure LSTM approach. Overall, their hybrid approach scored an accuracy of 94 percent. For comparison: The CNN model scored 79.35 percent and the LSTM model scored 76.21 percent accuracy. However, the hyperparameters used, as well as the epochs for training varied significantly. For example the LSTM model was trained for 15 epochs, while their proposed model trained for 18 epochs [1].

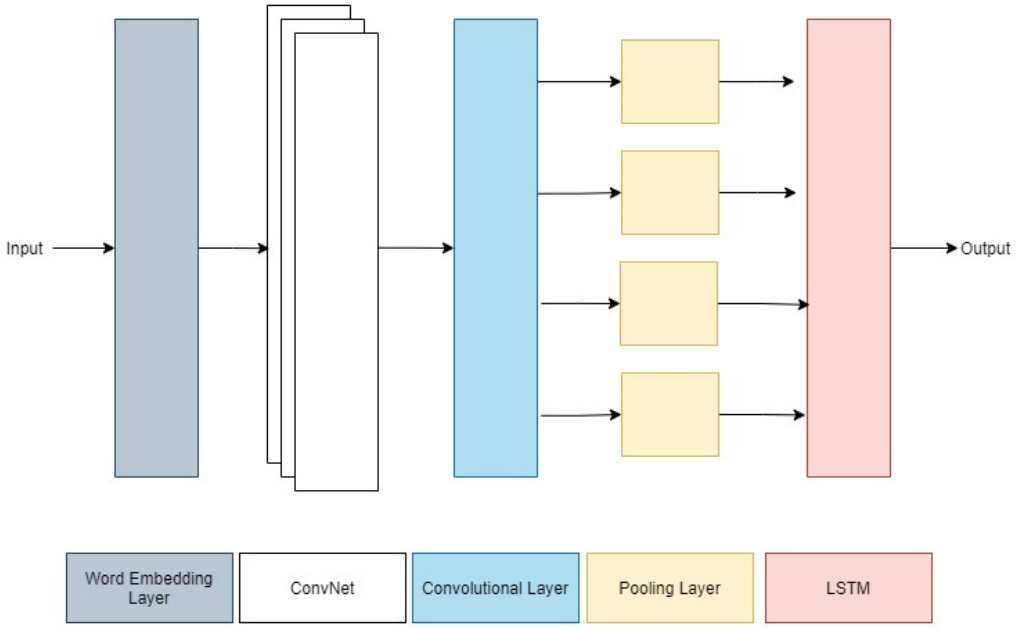

Figure 5: CNN-LSTM Hybrid architecture, Ankita at al. [1]

Rehman et al. proposed a similar approach. They also proposed a CNN-LSTM hybrid model and evaluated its performance. They trained and tested their models using two different datasets: IMDB movie reviews as well as Amazon movie reviews. In order to evaluate the performance of their architecture, they compared its performance to the performance of traditional DL approaches, such as CNN and LSTM. Their proposed model scored an accuracy of 91 percent on the IMDB dataset, which was a 4-8 percent improvement of f-measure compared to CNN and LSTM individually [20].

5 Implementation

This chapter describes the implementation of 3 deep learning models with different architectures for multi-class polarity classification. The models will be trained on the same data set with similar hyperparameters in order to compare their performance.

5.1 Hardware

The training process of the models was conducted on a system with the following specifications. The virtual machine used had 32 GB of RAM as well as 8 vCPU cores assigned. The central part of the setup is the GPU. In general, deep learning tasks heavily benefit from GPU usage in terms of training time. The GPU used in this project is a NVIDIA A30 Tensor Core GPU. This GPU is specifically designed for workloads like data analysis and deep learning. It also comes with 24 GB HBM2 Memory, which is plenty of space, even for bigger models [4].

5.2 Software

The VM used for this project runs on Ubuntu 22.04. The implementation itself is done in Python, version 3.10. While there are several machine learning frameworks available, this project is using the framework Tensorflow in the version 2.8.0. The implementations of the models in this project heavily relies on Keras, a high level deep learning API, which ships with Tensorflow [5]. Tensorflow itself can be described as a flexible and scalable software library for numerical computations. It was originally developed by researchers at Google and is one of the most popular deep learning frameworks in general. The core algorithms were developed in highly optimized C++ and further use NVIDIA's parallel computing API CUDA (Compute Unified Device Architecture). The Tensorflow API is available for several programming languages, although the API for Python is the most complete and stable one [19].

Tensorflow natively supports GPU usage. However, to enable the usage of a GPU with the Tensorflow framework on Ubuntu, additional software and drivers are necessary. First of all a compatible GPU is needed. Besides the correct drivers(in our case NVIDIA UNIX x86_64 version 515.65.01), CUDA(11.7) and cuDNN(8.5.0) drivers, as well as a suitable distribution of GCC (11.2.0) are essential. Although Tensorflow works very well out of the box, driver compatibility is a common source for errors. For data handling and preprocessing the libraries pandas and numpy were used. The data-visualization part of this work relies on the libraries matplotlib and sklearn.

An overview of all the software and libraries used for this thesis can be found in table 1.

Software/Library	Version
Ubuntu	22.04
NVIDIA UNIX x86_64	515.65.01
Python	3.10
CUDA	11.5
CUDNN	8.5.0
GCC	11.2.0
tensorflow	2.8.0
pandas	1.4.2
numpy	1.22.2
matplotlib	3.5.1
sklearn	1.0

Table 1: Software

5.3 Data set

In order to train the models, we use a subset of the Amazon Review data set from 2018. The complete data set consists of 233.1 million product reviews from the Amazon.com website. The subset we are using for training consists of 5.722.988 reviews of e-books from the Amazon Kindle Platform [18].

In order to ensure that the trained models are able to generalize well, e-books have been chosen for training. This is due to the fact that e-books in general can be about any topic, so the language in the reviews is not specific to a certain product category. In other words, the language of e-book reviews can be expected to be more general than the language in e.g. pet supplies (another subset of the Amazon review data set).
Another important factor that must be taken into consideration is the distribution of the different labels in the data set. It can be found in table 2.

Label	Reviews	Percent
1-Star	269522	4.7%
2-Star	220994	3.9%
3-Star	500582	8.7 %
4-Star	1252109	21.9 %
5-Star	3479781	60.8%

Table 2: Distribution data set

5.4 Data Preprocessing

The first step of the implementation is the preprocessing of the data. We need to choose a valid representation of the data in order to enable the model to learn from them.

Preprocessing The preprocessing part of this project relies on a Python library called pandas, which comes with a variety of data structures and tools for data manipulation and analysis [16]. The data set comes with several columns of metadata, which are not relevant for this project. For the training we are simply interested in the text of the actual review itself, combined with the rating(1-5 stars). The first step is to reduce the data set to the relevant columns, in our case "reviewText", which are the review texts themselves and "overall", which consist of integers between 1 and 5, equivalent to the star-based rating system Amazon is using.

After this first step, the data set is split into subsets for training, validation and testing. The subsets are further split: As a supervised learning approach as described in chapter 3 is used, the training data consist of values for the input(review text) as well as a prelabeled, corresponding output(rating). Input(x) and output(y) have to be preprocessed differently.

For the ratings(y-values), an approach called "one-hot" encoding was chosen. Although the integer representation of our ratings are suitable for the training process, the performance of deep learning models greatly benefits from a matrix-representation of the data. This is achieved with "one-hot" encoding. While there are a lot of ways to achieve this, a method from the Keras framework, namely keras.utils.to_categorical, was used. The method converts the resulting array is then converted to a numpy array, in order to feed it into the model.

Tokenization For the input values we chose word embeddings as a representation of the data. This is achieved by using the Tokenizer class from Keras. The Keras Tokenizer is used to vectorize a text corpus [5]. The Tokenizer is first initiated with parameters as follows:

```
Tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@
    [\\]^_`{|}~\t\n', lower=True, split=' ', char_level=
    False, oov_token="oovtoken")
```

Since we are only interested in the words themselves and their order in a given review, tokens are filtered out, which are not needed for the training process.

This functionality is part of the Tokenizer class and we simply specify the in “filter” parameter, what symbols to filter out. Furthermore, all the words are converted to lowercase (lower=True) and the texts are split into single words at the whitespace between them(split=’ ‘). Additionally, a token for words which are out of vocabulary is declared. This is done in the case the model needs to predict values for inputs which have words not present in the training set. These words are out of vocabulary and are therefore assigned with the oov_token value in order to avoid errors later on [5].

In the next step the internal vocabulary of the Tokenizer is updated. This is achieved with the class method "fit_on_texts()". The method gets a list of the texts as a parameter and then updates the internal vocabulary, where each unique word is assigned a unique integer. Afterwards sequences of integers based on the internal vocabulary are generated with the class method “text_to_sequences()” [5].

The last step in the preprocessing phase is to create word embeddings out of the integer sequences. We use the built-in method pad_sequences() to achieve this. The method in general creates a padding out of a specified value (default=0) to a given length. We call the method given the integer sequence as input and we further specify the maximum length a sequence can have [5].

Below we can see the whole process of preprocessing on the example of the input training data(x):

```
Tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@
    [\\]^_‘{|}~\t\n', lower=True, split=' ', char_level=
    False, oov_token="oovtoken")
Tokenizer.fit_on_texts(list_texts)
train_seq = Tokenizer.texts_to_sequences(train_x)
train_seq = pad_sequences(train_seq, maxlen=MAX_LEN)
```

5.5 Model Architecture

Within the Tensorflow framework, there are different methods to build a model. Since building a model from scratch is a very ambitious task, this project relies on Keras, a high level API within the framework which facilitates the implementation. In this particular case, the class "tf.keras.Sequential" was used. The purpose of said class is to group a linear stack of layers into a model (tf.keras.Model). The method add() connects a new layer to the

model. When all layers are added, the method compile() is called, which configures the model for the training process according to the given parameters. In this case a loss function, the optimizer and metrics for the output of the training process are declared. For the loss function, the only appropriate choice is the function categorical crossentropy. This function is suitable, if the training data has two or more labels and if the labels are one-hot encoded, which is the case as described in section 5.4 [5]. For the optimizer we chose Adam, described in detail in section 3.5.2. The first layer in all of the models is the embedding layer. The layer transforms the positive integers of the preprocessed training data into a dense vector of a fixed size [5]. The output layers of the three models are also identical. For the output layer a regular densely-connected layer with 6 units was used. 6 units are used, one for every label (1-5) plus an additional one (0), which is necessary, because the labels were not previously adjusted to range from 0-4. Further the activation function of the layer is specified in the parameters of the layer. We use the "softmax" function, mentioned in section 3.4.1. Overall the intentions of this work is to keep the model architectures simple and to use similar parameters in order to compare the performance of the different models. This leads to the problem, that the models are not very optimized, which negatively influences accuracy and performance overall. It is clear, that different model types require different hyperparameters and optimization. However, the primary goal of this work is not to implement the best performing models overall, but to compare model performance in terms sentiment analysis.

5.5.1 CNN

The implementation of the convolutional neural network has besides input and output layer 2 additional layers. We use a one-dimensional convolutional layer with the ReLU activation function, followed by a pooling layer. All in all, the model has 15,068,926 trainable parameters.

```
CNN = Sequential()
CNN.add(Embedding(input_dim=VOCAB_SIZE, output_dim=32,
    input_length=MAX_LEN))
CNN.add(Conv1D(filters=NUM_FILTERS, kernel_size=
    KERNEL_SIZE, activation="relu"))
CNN.add(GlobalMaxPooling1D())
CNN.add(Dense(6, activation="softmax"))
CNN.compile(loss='categorical_crossentropy', optimizer=
    'adam', metrics=['accuracy'])
CNN.summary()
```

5.5.2 RNN

For the implementation of of the recurrent neural network only one additional layer is needed. For this RNN implementation, a LSTM-layer is used, which is described in more detail in section 3.5.4. In the LSTM layer we have to specify parameters. 128 units are used and a dropout (fraction of the units to drop for the linear transformation of the inputs) of 0.01 is declared. The LSTM-layer uses the "tahn" activation function. In case of a recurrent activation it uses the sigmoid activation function. The model has 16,197,542 trainable parameters.

```
RNN = Sequential()
RNN.add(Embedding(input_dim=VOCAB_SIZE, output_dim=32,
    input_length=MAX_LEN))
RNN.add(LSTM(1024, dropout=0.01))
RNN.add(Dense(6, activation="softmax"))
RNN.compile(loss='categorical_crossentropy', optimizer=
    'adam', metrics=['accuracy'])
RNN.summary()
```

5.5.3 Hybrid

The implementation of the CNN-RNN Hybrid model relies on proposed model architectures mentioned in section 4, such as the one proposed by Ankita et al: In practice, the convolutional layer and the pooling layer from the CNN-implementation are combined with an LSTM layer. This model has a total of 23,363,470 trainable parameters.

```
HYBRID = Sequential()
HYBRID.add(Embedding(input_dim=VOCAB_SIZE, output_dim
    =32, input_length=MAX_LEN))
HYBRID.add(Conv1D(filters= NUM_FILTERS, kernel_size=
    KERNEL_SIZE, activation="relu"))
HYBRID.add(GlobalMaxPooling1D(keepdims=True))
HYBRID.add(LSTM(1024, dropout=0.01))
HYBRID.add(Dense(6, activation="softmax"))
HYBRID.compile(loss='categorical_crossentropy',
    optimizer='adam', metrics=['accuracy'])
HYBRID.summary()
```

5.6 Hyperparameters

MAX_LEN This parameters controls the maximum length a review can have. For this work the parameter is set to 300. This means the length of a review which is going to be processed by the model will have exactly a length of 300 words. If the review should be longer than 300 words, it will be shortened. If it is shorter, the review, which is represented as an array of integers as described in section 5.4, is padded with zeroes. In the data set used for this thesis 1984893 (34.8%) of the reviews are longer than 300 words.

BATCH_SIZE The training process in the Tensorflow framework is usually done in batches. The value batch_size controls how many reviews are processed, before the models internal parameters are updated [5]. For this work a batch size of 128 is used.

EPOCHS The hyperparameter epochs controls, how often the training data set is presented to the model [5]. In general, the training accuracy of the model increases with every additional epoch. On the other hand, if the number of epochs in the training process is too high, the model is likely overfitting. For the sample implementation we train the different models for 7 epochs.

SHUFFLE This Boolean parameter controls, whether the training data is shuffled before each epoch. For this work we stick to the Tensorflow default, which is "True" [5].

NUM_FILTERS In those models, which have convolutional layers, this parameter determines the number of filters in the convolutional layer [5]. For this project 1000 filters are used.

KERNEL_SIZE The size of the kernel is only relevant for the models with convolutional layers. The parameter specifies the length of the convolution window [5]. The kernel size in this work is 100.

VOCAB_SIZE This parameter is equal to the size of the vocabulary of the training data. In other words it is the number of unique words in the training data set. The number depends on how much training data is used and it is taken directly from the internal vocabulary of the Tokenizer mentioned in section 5.4.

6 Results

In order to start the training process the method fit() is called on every model. Additionally we declare the training data (train_seq), labels (train_y), as well as validation data(val_seq) and the corresponding labels(val_y). Additional parameters are explained in detail in section 5.6.

```
CNN.fit(train_seq, train_y, validation_data=(val_seq,
    val_y), batch_size=BATCH_SIZE, shuffle=SHUFFLE,
    epochs=EPOCHS)
RNN.fit(train_seq, train_y, validation_data=(val_seq,
    val_y), batch_size=BATCH_SIZE, shuffle=SHUFFLE,
    epochs=EPOCHS)
HYBRID.fit(train_seq, train_y, validation_data=(val_seq
    , val_y), batch_size=BATCH_SIZE, shuffle=SHUFFLE,
    epochs=EPOCHS)
```

6.1 Training Accuracy

The training accuracy after 7 epochs can be found in the table below. The training accuracy is calculated by Tensorflow during the training process.

Model	Accuracy
CNN	81.41%
RNN	77.26%
HYBRID	82.32%

Table 3: Training Accuracy

6.2 Validation Accuracy

The validation accuracy after 7 epochs can be found in the table below.

Model	Accuracy
CNN	69.20 %
RNN	70.71 %
HYBRID	69.25 %

Table 4: Validation Accuracy

6.3 Testing

For model testing a random sample of 10000 reviews was chosen from the same data set, which the models have not seen before (The testing data is not part of training or validation data set). The trained models predict the label for every review(1-5). The results are displayed in a so-called confusion matrix. The y-axis shows the true label, taken from the data set itself. The x-axis shows the predicted label of each model.

Model	Accuracy
CNN	68.14 %
RNN	69.17 %
HYBRID	68,52%

Table 5: Test accuracy

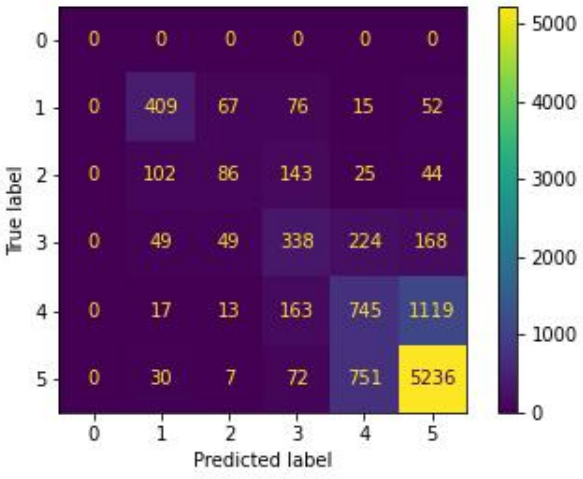

Figure 6: Confusion Matrix CNN implementation

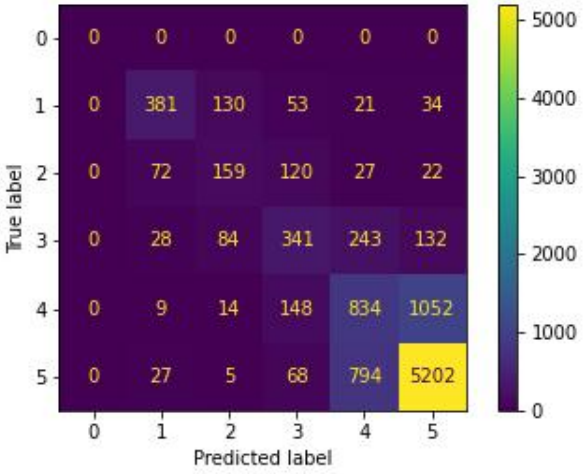

Figure 7: Confusion Matrix RNN implementation

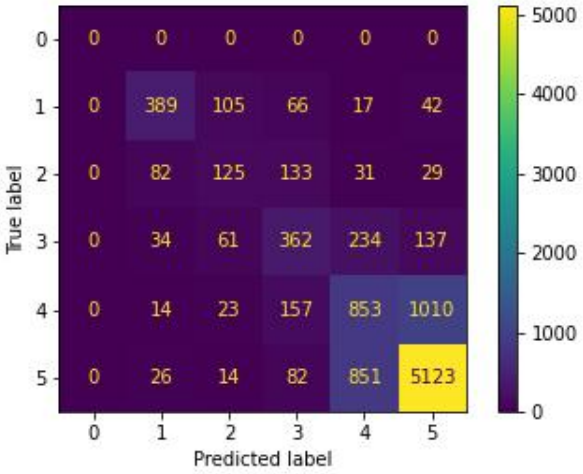

Figure 8: Confusion Matrix HYRBID implementation

Label	Precision	Recall	F1-score	support
1	0.66	0.67	0.67	607
2	0.21	0.39	0.28	222
3	0.41	0.43	0.42	792
4	0.36	0.42	0.39	1760
5	0.86	0.79	0.82	6619

Table 6: Scores CNN

Label	Precision	Recall	F1-score	support
1	0.62	0.74	0.67	517
2	0.40	0.41	0.40	392
3	0.41	0.47	0.44	730
4	0.41	0.43	0.42	1919
5	0.85	0.81	0.83	6442

Table 7: Scores RNN

Label	Precision	Recall	F1-score	support
1	0.63	0.71	0.67	545
2	0.31	0.38	0.34	328
3	0.44	0.45	0.44	800
4	0.41	0.43	0.42	1986
5	0.84	0.81	0.82	6341

Table 8: Scores Hybrid

6.4 Experiments

In this section some experiments are conducted. Some parameters of the model are varied, to see, how the accuracy of the different model types are affected.

6.4.1 Increased Training Data

To evaluate, how an increased amount of training data impacts the accuracy of the different models, the amount of training data was increased to 3.000.000 reviews. The validation data set remains at 2.000.000 reviews. All the other parameters are equal to the parameters in the reference implementation. The results can be taken from the tables below. The results are

compared to the results of the reference implementation in section 6.

Model	Training Accuracy	Difference
CNN3/7	79.59%	-1.82%
RNN3/7	79.23%	+1.97%
HYBRID3/7	80.22%	-2.1%

Table 9: Training Accuracy, 3M, 7 Epochs

Model	Validation Accuracy	Difference
CNN3/7	69.03%	-0.17%
RNN3/7	68.86%	-1.85%
HYBRID3/7	69.45%	+0.2%

Table 10: Validation Accuracy, 3M, 7 Epochs

Model	Test Accuracy	Difference
CNN3/7	50,96%	-17.18%
RNN3/7	46,84%	-22.33%
HYBRID3/7	48,69%	-19.83 %

Table 11: Test Accuracy, 3M, 7 Epochs

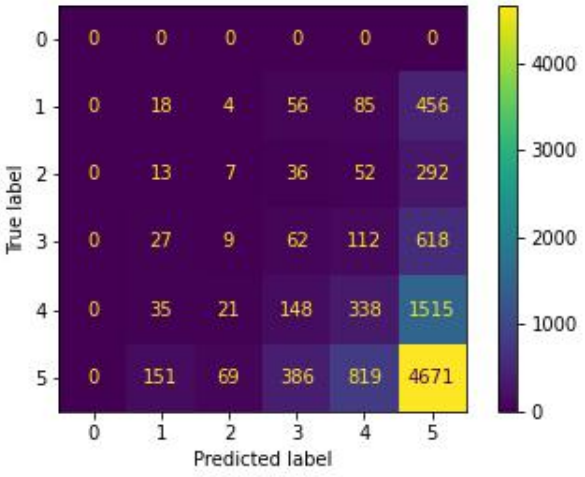

Figure 9: Confusion Matrix CNN, 3M, 7 Epochs

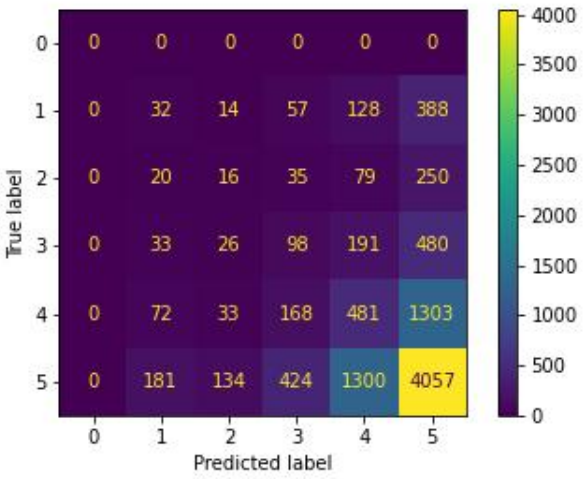

Figure 10: Confusion Matrix RNN, 3M, 7 Epochs

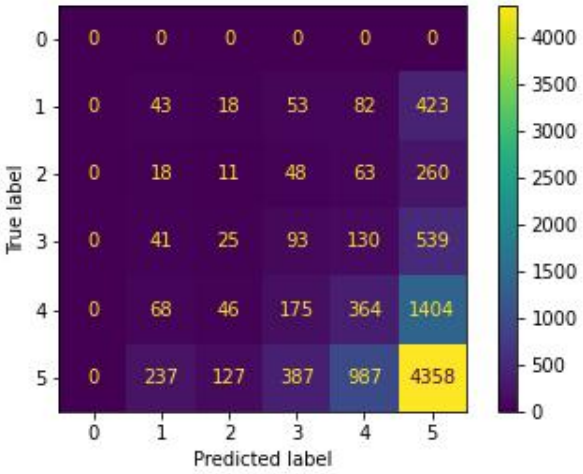

Figure 11: Confusion Matrix HYBRID, 3M, 7 Epochs

6.4.2 Increased Epochs

To answer the question how an increase in epochs will affect the different model types, the models have been trained an additional time with 2.000.000 reviews in the training data set. The epochs were increased from 7 to 12.

Model	Training Accuracy	Difference
CNN2/12	85.47%	+4.06 %
RNN2/12	83.43%	+6.17%
HYBRID2/12	87.13%	+4.81%

Table 12: Training Accuracy, 2M, 12 Epochs

Model	Validation Accuracy	Difference
CNN2/12	67.78%	-1.42%
RNN2/12	68.62%	-2.09%
HYBRID2/12	68.66%	-0.57%

Table 13: Validation Accuracy, 2M, 12 Epochs

Model	Test Accuracy	Difference
CNN2/12	66,8%	-2.4%
RNN2/12	67,29%	-1.88%
HYBRID2/12	67,11%	-1.41%

Table 14: Test Accuracy, 2M, 12 Epochs

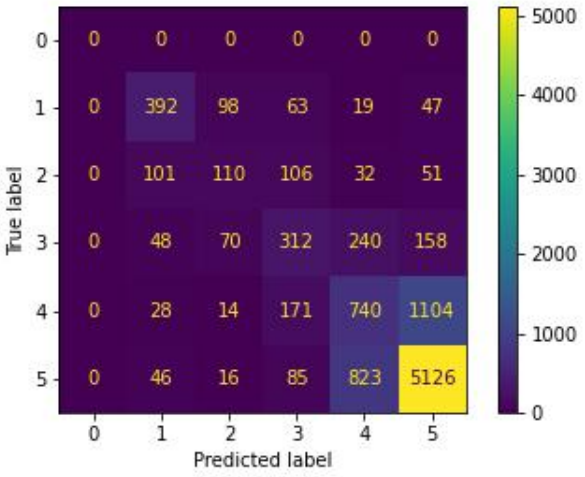

Figure 12: Confusion Matrix CNN, 2M, 12 Epochs

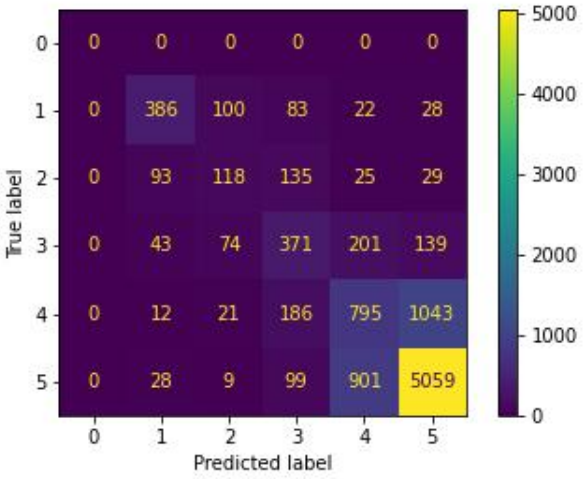

Figure 13: Confusion Matrix RNN, 2M, 12 Epochs

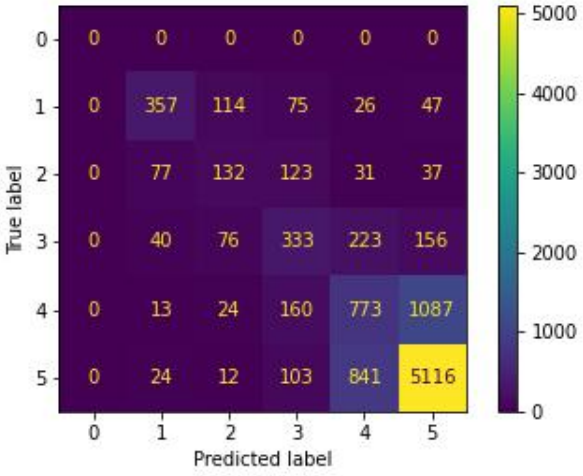

Figure 14: Confusion Matrix Hybrid, 2M, 12 Epochs

7 Discussion and Limitations

The corresponding code for this work is publicly available on github at https://github.com/ehaue1s/ba-thesis. In general the accuracy of the models in terms of multiclass classification is limited. The reference implementations

score around 70% accuracy on the test data. This is also inline with the validation accuracy of the models. This means in general, that the different models guess the right class about 70 percent of the time. However, at this point we have to address several problems.

7.1 Optimization

The models in general are not very optimized. Considering the usage of similar/identical hyperparameters for the different model types to ensure comparability, it is only logical, that the performance of each individual model is not ideal. As we can see from the test accuracy tables section 6.3, 6.4.2 and 6.4.1, the individual models respond differently to a variation of hyperparameters.

7.2 Overfitting

While it is impossible to avoid overfitting completely. Common sources of overfitting are noisy training data, limited size of the training data or constraints of the used algorithms [28]. The results of the experiments in section 6.4.2 show increased training accuracy combined with a slightly lower validation and test accuracy. This could already be a sign of overfitting: the model is more accurate on the training data but its ability to generalize decreased.

7.3 Underfitting

The experiment 6.4.1 shows, that an increase in training data is not always desirable. Although the training data was increased by 50%, the test accuracy of the models decreased between 17.18% and 22.33%. This is a very significant decrease in performance. In the tables in the Appendix A, we can see that the different models all have the same problem: Although the precision for the label 5 is high (CNN 0.77, RNN 0.67, HYBRID 0.71) the precision for all the other labels is miserable. The overall low training accuracy combined with the very bad test accuracy indicates, that the models are underfitting. The results indicate, that 7 epochs are too little for such a huge increase in training data.

7.4 Data distribution

Unbalanced data sets in general are very problematic for supervised learning. If there are class imbalances in the training data, models are over-classifying the majority group [12]. If we take a look at how the different labels are

distributed in the data set, we can see that 60.8% of the reviews are 5-star reviews, while only 4.7% of the data consists of 1-star reviews. 2-3 star ratings are also underrepresented. Looking at the evaluation of the results in the tables 6, 7 and 8, we can clearly see that the precision of the model is very high for the labels 1 and 5, but very low for the labels 2-3. This confirms the negative impact of the imbalanced data set on the performance of the models. The models are very good at classifying 5 star reviews but lack performance on the other labels.

7.5 Multiclass classification

To get better results in binary and multiclass classification while using imbalanced data sets, all the classes have to be well represented within the data set and come from non-overlapping distributions [14]. While the first condition might be true for this work, the second one arguably does not hold. Although the data is labeled from 1-5, the individual classes can not be strictly distinguished. A 1-star review on Amazon is most of the time very different to a 5-star review in terms of language. A 2 and 3-star review on the other hand might be indistinguishable from each other. These factors also negatively impact the ability of the models to generalize. All of the models in this work have problems to classify 2-4 star reviews. This indicates, that for this type of problem a binary classification would be overall more useful and would perform better overall. This could be accomplished quite easily: First, the labels in the data set would be scaled from 1-5 to positive/negative or 0/1. Next, the output layers of the models are changes to 2 output classes and the loss function would be changed to binary crossentropy.

7.6 Discussion

With the problems described in the previous sections in mind, lets take a look at the overall performance of the models. Neither an increase in training data, nor an increase in epochs had a positive impact on the test accuracy of the models. The models from the reference implementation in section 5 performed the best overall. However, the difference in performance between the model types is too small to draw any meaningful conclusions from it. We can conclude, that all of the model types described in this work are useful for the task of sentiment analysis with overall high performance. The performance could be further improved by either using a data set with balanced classes, by implementing the models for binary classification or by using hyperparameter-tuning.

8 Summary

This work shows the implementation process of 3 different deep learning models for document level sentiment classification. The underlying research question of this thesis is "How do different deep learning models, namely convolutional neural networks and recurrent neural networks perform at sentiment analysis?". In order to answer said question, a convolutional neural network, a recurrent neural network, as well as a hybrid solution were successfully implemented and trained using Amazon review data [18]. Data suggests, that the models classify the correct sentiment, in this case the equivalent of the star rating on Amazon.com, in around 70% of the cases. Furthermore, consistent performance over all the different model types can be observed.

Model	Accuracy
CNN	68.14 %
RNN	69.17 %
HYBRID	68,52%

Table 15: Test accuracy, reference implementation

Further analysis shows, that the performance of the models suffers from imbalanced classes in the data set and lack of optimization. Based on that, several several suggestions are made to further increase the overall performance of the different models. This work, as well as the corresponding code can be found on https://github.com/ehaue1s/ba-thesis.

A Appendix

A.1 Test Scores Experiments

Label	Precision	Recall	F1-score	support
1	0.03	0.07	0.04	244
2	0.02	0.06	0.03	110
3	0.07	0.09	0.08	688
4	0.16	0.24	0.20	1406
5	0.77	0.62	0.68	7552

Table 16: Score CNN, 3M, 7 Epochs

Label	Precision	Recall	F1-score	support
1	0.05	0.09	0.07	338
2	0.04	0.07	0.05	223
3	0.12	0.13	0.12	782
4	0.23	0.22	0.23	2179
5	0.67	0.63	0.65	6478

Table 17: Score RNN, 3M, 7 Epochs

Label	Precision	Recall	F1-score	support
1	0.07	0.11	0.08	407
2	0.03	0.05	0.04	227
3	0.11	0.12	0.12	756
4	0.18	0.22	0.20	1626
5	0.71	0.62	0.67	6984

Table 18: Score HYBRID, 3M, 7 Epochs

Label	Precision	Recall	F1-score	support
1	0.63	0.64	0.64	615
2	0.28	0.36	0.31	308
3	0.38	0.42	0.40	737
4	0.36	0.40	0.38	1854
5	0.84	0.79	0.81	6486

Table 19: Score CNN, 2M, 12 Epochs

Label	Precision	Recall	F1-score	support
1	0.62	0.69	0.65	562
2	0.29	0.37	0.33	322
3	0.45	0.42	0.44	874
4	0.39	0.41	0.40	1944
5	0.83	0.80	0.82	6298

Table 20: Score RNN, 2M, 12 Epochs

Label	Precision	Recall	F1-score	support
1	0.58	0.70	0.63	511
2	0.33	0.37	0.35	358
3	0.40	0.42	0.41	794
4	0.38	0.41	0.39	1894
5	0.84	0.79	0.82	6443

Table 21: Score HYBRID, 2M, 12 Epochs

References

[1] Ankita, Shalli Rani, Ali Kashif Bashir, Adi Alhudhaif, Deepika Koundal, and Emine Selda Gunduz. An efficient CNN-LSTM model for sentiment detection in #BlackLivesMatter. *Expert Systems with Applications*, 193:116256, 2022.

[2] Richard Baskerville, Richard, Jan Pries-Heje, Jan, John Venable, and John. Soft design science methodology. 01 2009.

[3] Rene Y. Choi, Aaron S. Coyner, Jayashree Kalpathy-Cramer, Michael F. Chiang, and J. Peter Campbell. Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational Vision Science & Technology*, 9(2):14–14, February 2020.

[4] Nvidia Corporation. Datasheet: Nvidia a30 tensor core gpu, 2022.

[5] Martin Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[6] Ronen Feldman. Techniques and applications for sentiment analysis. *Commun. ACM*, 56(4):82–89, apr 2013.

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

[8] Olivier Habimana, Yuhua Li, Ruixuan Li, Xiwu Gu, and Ge Yu. Sentiment analysis using deep learning approaches: an overview. *Science China Information Sciences*, 63(1):111102, December 2019.

[9] Julia Hirschberg and Christopher D. Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015. _eprint: https://www.science.org/doi/pdf/10.1126/science.aaa8685.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, December 1997.

[11] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, September 2021.

[12] Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, March 2019.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[14] Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[16] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

[17] Yelena Mejova. Sentiment analysis: An overview. 2009.

[18] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 188–197, 2019.

[19] Bo Pang, Erik Nijkamp, and Ying Nian Wu. Deep Learning With TensorFlow: A Review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248, 2020. _eprint: https://doi.org/10.3102/1076998619872761.

[20] Anwar Ur Rehman, Ahmad Kamran Malik, Basit Raza, and Waqar Ali. A Hybrid CNN-LSTM Model for Improving Accuracy of Movie Reviews Sentiment Analysis. *Multimedia Tools and Applications*, 78(18):26597–26613, September 2019.

[21] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[22] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[23] Pei Wang. On defining artificial intelligence. *Journal of Artificial General Intelligence*, 10(2):1–37, 2019.

[24] Mayur Wankhade, Annavarapu Chandra Sekhara Rao, and Chaitanya Kulkarni. A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, February 2022.

[25] Janyce Wiebe, Theresa Wilson, Rebecca Bruce, Matthew Bell, and Melanie Martin. Learning Subjective Language. *Computational Linguistics*, 30:277–308, September 2004.

[26] Martin Wöllmer, Florian Eyben, Björn Schuller, Ellen Douglas-Cowie, and Roddy Cowie. Data-driven clustering in emotional space for affect recognition using discriminatively trained LSTM networks. pages 1595–1598, September 2009.

[27] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing, 2017.

[28] Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168(2):022022, feb 2019.