SCHOOL OF
ECONOMICS AND
MANAGEMENT

Sentiment Analysis for Talent Attraction

Enhancing Sentiment Analysis Performance by Improving Accuracy of Word Embeddings Using a Transformer-Based Approach

by

Jelena Zec

June 2023

Master's Programme in 2022

Supervisor: Najmeh Abiri

Abstract

The reputation of a company on employer review platforms can have a significant impact on its ability to attract talented workers. Companies use sentiment analysis to learn how their employer brand is perceived online. Furthermore, sentiment analysis can detect strengths and weaknesses in their employer brand, indicating which areas need improvement.

The proposed methods for improving word embeddings for sentiment analysis commonly involve combining several pre-trained word embeddings, or concatenating vector representations of non-textual elements (e.g., emojis and images) to word embeddings. These methods involve training complex neural networks, which is usually computationally expensive.

This thesis investigates if adding features prior to tokenization, instead of concatenating embeddings, increases the accuracy of word embeddings, thereby improving the results of the fine-tuned BERT model for classifying sentiment of employer reviews on the online platform Glassdoor. It also investigates the impact of the BERT Next Sentence Prediction objective on the models' ability to learn more accurate word embeddings.

Testing three different models and comparing their performance indicates that the suggested approach can improve the model's accuracy. However, additional research is needed to investigate the impact of the chosen features on the observed results. The study hasn't found enough evidence that addition of the Next Sentence Prediction objective results in higher accuracy of the model, but it shows that it significantly improves model ability to understand the sentiment of the reviews.

Acknowledgements

I would like to thank my supervisor Najmeh Abiri for providing me with great guidance and support throughout this process. Thank you for your feedback and ideas, which were crucial for successfully completing this work.

Also, I would like to thank Joakim Westerlund, the MSc in Data Analytics and Business Economics program director, whose advice and support were of great help for choosing and defining the research topic.

Table of Contents

1 Introduction 7
1.1 Background 7
1.2 Aim and Objectives 8
1.3 Research Purpose 8
1.4 Delimitations 10
2 Theoretical Background 11
2.1 Sentiment Analysis 11
2.2 Traditional Methods for Sentiment Analysis 12
2.2.1 Lexicon-Based Methods 12
2.2.2 Machine Learning Methods 12
2.3 Deep Learning Methods 13
2.3.1 Word Embeddings 13
2.3.2 Recurrent Neural Networks 15
2.3.3 Long Short-Term Memory (LSTM) 16
2.3.4 Bidirectional Recurrent Neural Networks 17
2.3.5 Convolutional Neural Networks 18
2.3.6 Attention 20
2.4 Transformer 22
2.4.1 Encoder 24
2.4.2 Decoder 28
2.5 BERT 30
2.5.1 Input/Output Representation 30
2.5.2 Pre-Training 31
2.5.3 Fine-Tuning and Feature-Based Approach 33
2.5.4 Model Output 34
3 Data 36
3.1 Dataset Exploration 37
4 Methodology 41
4.1 Research Design 41
4.2 Data Preparation 41
4.3 Model 44
4.3.1 Tokenization 44
4.3.2 Model Architecture 47

4.3.3 Regularization and Optimization 48
5 Evaluation 49
5.1 Training and Evaluation 49
5.2 Results 51
5.3 Discussion 55
5.4 Limitations 57
5.5 Future Work 57
6 Conclusions 58
References 59
Appendix A 66
Appendix B 67
Appendix C 69

List of Tables

Table 1 Glassdoor Job Review dataset - variables overview 36
Table 2 Examples of reviews 38
Table 3 Rulesets for assigning the sentiment 42
Table 4 Evaluation results 51
Table 5 Model classification report 52

List of Figures

Figure 1 ELMo pre-trained model 14
Figure 2 Recurrent neural network for sentiment analysis 15
Figure 3 Structure of the LSTM cell 17
Figure 4 Architecture of bidirectional RNN 18
Figure 5 Example of convolution: input, output, and filter dimensions 3x4 19
Figure 6 Example of convolution: input, output. Number of filters is 3, with padding and average pooling through time 20
Figure 7 Sequence-to-sequence with attention 22
Figure 8 Transformer model architecture 23
Figure 9 Self-attention (Scaled Dot-Product attention) 24
Figure 10 Multi-head attention 25
Figure 11 An example of multi-head attention mechanism 26
Figure 12 Surface of the loss function without and with "shortcut connections" 27
Figure 13 Masking in self-attention 29
Figure 14 BERT input representation 31
Figure 15 Architecture of the BERT pre-trained model 32
Figure 16 Masking example 33
Figure 17 Example of training dataset used for NSP 33
Figure 18 Fine-tuning BERT 34
Figure 19 Dimensions of the BERT model last hidden state 35
Figure 20 Distribution of overall ratings 37
Figure 21 Ratings of twenty companies with largest number of reviews 37
Figure 22 Distribution of rating over twenty most common job titles 38
Figure 23 Length of text of "pros" and "cons" across overall rating values 40
Figure 24 Distribution of the sentiment class 43
Figure 25 BERT WordPiece tokenizer output 44
Figure 26 Length of reviews across sentiment values 45
Figure 27 The final model architecture 47
Figure 28 Model 1 - loss and weighted F1 score curve 50
Figure 29 Model 2 - loss and weighted F1 score curve 50
Figure 30 Model 3 - loss and weighted F1 score curve 51
Figure 31 Model 1 - Confusion matrix, precision-recall curve 53
Figure 32 Model 2 - Confusion matrix, precision-recall curve 53
Figure 33 Model 3 - Confusion matrix, precision-recall curve 54
Figure 34 Summary of the final model 66

1 Introduction

This chapter briefly discusses the significance of attracting a talented workforce, and the challenges that companies face in this process. It also provides a motivation for using sentiment analysis as an auxiliary tool for a more successful outcome of this process. Furthermore, it outlines the aim, objective, and purpose of the thesis. Additionally, the delimitations and the outline of the thesis are given.

1.1 Background

In the past three decades, globalization, the emergence of new technologies, and continued innovation have become crucial for companies to compete and stay relevant in the markets. "[C]ompetitive advantage rests on making more productive use of inputs" (Porter, 1998, p.78), as well as product and service innovation. This has led to the rise of the knowledge-based economy where competencies, skills and knowledge of the employees have become crucial for company success. As the demand for a talented workforce has grown, attracting highly competent employees has been an ongoing challenge (Schuler, Jackson & Tarique, 2011; McKinsley&Company, 2022). Consequently, companies have started to compete for the best employees, starting a "war for talent" (Beechler & Woodward, 2009).

Many studies have shown that reputation significantly influences a company's ability to attract talent (Cable & Turban, 2003; Schaarschmidt, Walsh & Ivens, 2021). Furthermore, building an image as a good employer can have long term positive effects. "[R]eputation is one of the few resources that can give firms a sustainable competitive advantage, because it is viewed as a non-tradable, non-substitutable, non-imitable, resource that can be managed" (Ferris, Perrewé, Ranft, Zinko, Stoner, Brouer & Laird, 2007, p.119).

Today, many job seekers research employers prior to deciding whether to apply for a job. On employment platforms such as Glassdoor and Indeed, they can find reviews of companies provided by current and former employees.

> [They] can rely on the user-generated reports to reduce the information asymmetry that typically marks job applications, because they gain first-hand insights from others with practical work experience with the organization (Schaarschmidt, Walsh & Ivens, 2021 p.2).

The increased usage of these platforms has made it crucial for companies to have a good reputation on them (Schaarschmidt, Walsh & Ivens, 2021).

As the impact of a company's reputation on the online employer platforms has started to affect its ability to attract talent, many have become interested "[in promoting] themselves as an employer of choice and attractive workplace" (Kashive, Khanna & Bharthi, 2020, p.94). Today, sentiment analysis is used to gain in-depth knowledge about the company's reputation, identify the area where it needs to improve as an employer, and compare with the competition.

Sentiment analysis is a Natural Language Processing (NLP) technique used for analyzing "people's opinions, sentiments, evaluations, attitudes, moods, and emotions" (Liu, 2017, p.2). With the increase in usage of social media and online platforms, it has become an important tool for businesses to make better decisions (Alessia, Ferri, Grifoni & Guzzo, 2015).

While sentiment analysis of opinions expressed on social media, forums, and customer reviews is increasing in popularity, it faces the challenge of the "constant evolution of the language used online in user-generated content" (Pozzi et.al., 2017, p.9). Other challenges relate to recognition of figurative speech (e.g., sarcasm and irony), negotiation handling (e.g., words such as "nor" or "neither"), and spam detection (e.g., fake reviews)—all of which have negative impacts on the accuracy of analysis (Birjali, Kasri and Beni-Hssane, 2021).

1.2 Aim and Objectives

The aim of this thesis is to investigate whether incorporating information about the job title and company name into word embeddings can enhance the performance of sentiment analysis of employee reviews on the online platform Glassdoor.

To assess the impact that encoding this information in the word embeddings has on sentiment analysis performance, three pre-trained BERT models are fine-tuned, and their performances are compared. First, based on the overall rating, a sentiment—"positive", "neutral" and "negative"— is assigned to all reviews. The first model is fine-tuned only using the employee's review. The second model is fine-tuned using the employee's review, job title, and company name, which are concatenated into a single sequence. The third model is fine-tuned using the sequence pairs, where the first sequence is the employee's review, and the second sequence is a concatenation of the job title and company name.

1.3 Research Purpose

The word embeddings capture words meanings by projecting them into the d-dimensional semantic features space. Due to their vital impact on the performance of the different *Natural Language Processing* (NLP) applications (e.g., sentiment analysis, text classification, recommendation systems) this research field has gained a lot of attention in the past decade, and several pre-trained language models have been developed. These models are trained on large-scale corpora, and used to produce contextualized word embeddings which serve as an

input to a downstream task. The resulting embeddings are commonly fine-tuned to capture domain specific word features.

Research has shown that improving accuracy of, or adding features to, pre-trained word embeddings can improve model performance in various NLP tasks (Rezaeinia, Rahmani, Ghodsi & Veisi, 2019; Wang, Jiang, Bach, Wang, Huang, Huang & Tu, 2020). One popular approach in many NLP tasks is to concatenate the word embeddings produced by several pre-trained models (Rezaeinia et al., 2019; Wang et al., 2020). In text classification tasks, categorical or numerical features from the dataset are added to provide additional context. This is done by concatenating feature vector representations to word embeddings. In sentiment analysis, knowledge (Sinoara, Camacho-Collados, Rossi, Navigli & Rezende, 2019) and sentiment enhancement (Li, Li, Du, Fan & Chen, 2022) models have been proposed. Other aspects of social media posts, such as emojis (Liu, Fang, Lin, Cai, Tan, Liu & Lu, 2021) and images (Graesser, Gupta, Sharma & Bakhturina, 2017), have also been used as additional features to the word embeddings (Liu, Fang, Lin, Cai, Tan, Liu & Lu, 2021). However, all these approaches involve building complex models that are computationally expensive to train.

The purpose of this thesis is to explore the following:

- Does adding features (e.g., job title and company name) prior to tokenization produce more accurate word embeddings, thereby improving the results of sentiment analysis of employees' reviews?
- Does the Next Sentence Prediction task help the BERT model to learn more accurate word embeddings?

Two approaches are tested to try to answer these research questions. In the first approach, reviews and features are concatenated into a single sequence prior to tokenization. In the second approach, the reviews and the concatenated additional features make out sequence pairs, and the BERT model uses the Next Sentence Prediction classification task to acquire additional knowledge regarding the relationship between an employee's review and the added features. To my knowledge, no previous research has investigated the impact of this method on the sentiment analysis of employees' reviews.

When reviewing a company, employees most often write about the aspects of the workplace connected to the company culture, job conditions (salary and benefits), etc. It is reasonable to assume that people who work in similar positions within one company, or who work in similar companies, will praise and criticize similar aspects of the workplace. For example, if salaries in a company are low, the assumption is that most of the reviewers will mention that and give the company a lower rating. Therefore, including information about the job title and the company name prior to tokenization could help the model to better understand the context in which e.g., the word "salary" has a positive, neutral, or negative sentiment, and consequently improve the accuracy of the model.

1.4 Delimitations

The analysis is conducted on the Glassdoor Job Reviews dataset (DG, 2021). Due to the limited computational capability of the hardware used for training neural networks, and the limited time to conduct the research, the focus of the thesis is not to attain the highest accuracy of the models used. Rather, the aim is to explore if incorporating the additional information in the proposed manner can enhance the accuracy of sentiment analysis.

2 Theoretical Background

This chapter provides an overview of the techniques and the main concepts necessary to understand used models and the outcome of the research. Sections 2.1 through 2.3 provide an overview of the methods used for sentiment analysis, with the intention of providing the reader with a better understanding of how the model used in this thesis compares to other ones in the field. Section 2.4 gives an in-depth explanation of the Transformer model architecture, which was used to develop the BERT language model. Section 2.5 describes the BERT model, which was used in the research.

2.1 Sentiment Analysis

Sentiment analysis is a field of Natural Language Processing which involves the identification and extraction of subjective information from textual data. Liu and Zang (2012) describe sentiment analysis as a "computational study of people's opinions, appraisals, attitudes, and emotions toward entities, individuals, issues, events, topics and their attributes." The field gained significant attention in the mid-2000s due to the need to analyze vast amounts of unstructured text data available on the internet, including social media platforms, online reviews, and customer feedback. It has found applications in various fields, including finance, national security, politics, healthcare, marketing, and customer service. Businesses use sentiment analysis for example to understand customer opinions about their products, track brand sentiment or predict the development of financial markets.

Sentiment analysis can be regarded as a classification task as it classifies text into categories (e.g., positive, neutral, or negative) (Zhang, Lipton, Li & Smola, 2021). It can be conducted on three levels: document, sentence, or aspect level.

Document-level sentiment analysis determines the sentiment of the document as a whole. The disadvantage of this approach is that it "does not consider different sentences and aspects that a document may contain." (Habimana, Li, Li, Gu & Yu, 2020, p.3). For example, a review can contain both positive and negative aspects: "Overall the hotel is ok. The location is good, and the staff is friendly. However, they don't offer vegan food and the Wi-Fi is bad." Sentiment analysis at the document-level might classify this review as overall positive, missing the negative feelings toward the lack of vegan food and bad Wi-Fi.

Sentence-level sentiment analysis attempts to address this issue, by determining the sentiment of each single sentence. However, this approach suffers from a similar limitation because "one sentence may contain multiple entities with different aspects" (Habimana et al. 2020, p.3). For example, "The staff is very friendly, but the Wi-Fi is bad."

Aspect-level sentiment analysis "aims to find sentiments with respect to the specific aspects of entities" (Birjali, Kasri & Beni-Hssane 2021, p.2). In the above given example, it will capture the positive sentiment of the aspect "staff" and the negative sentiment of the aspect "Wi-Fi".

2.2 Traditional Methods for Sentiment Analysis

Traditional methods for analyzing sentiment are categorized into two groups: lexicon-based methods and machine learning methods.

2.2.1 Lexicon-Based Methods

Lexicon-based methods assign sentiment scores to the words in a sentence using pre-trained lexicons (e.g., SentiWordNet, SenticNet) (Habimana et al. 2020), which are then aggregated into a single score (Taboada, Brooke, Tofiloski, Voll & Stede, 2011). A drawback of this approach is that it assumes that the presence of more positive words in a sentence always indicates a positive sentiment sentence, which may not always be true (Kannan, Karuppusamy, Nedunchezhian, Venkateshan, Wang, Bojja & Kejariwal, 2016). Another limitation is that a word will always be assigned the same sentiment regardless of the context it appears in. For example, the adjective "small" in the sentence "My salary is small" and "The damage was very small" will be assigned the same sentiment, despite it being negative in the first sentence, and positive in the second. However, an advantage of these methods is that they don't require a training dataset and are computationally inexpensive to implement.

Lexicons are commonly created using a dictionary-based or corpus-based approach. In a dictionary-based approach, all words in a dictionary (e.g., WordNet) with similar meanings will be assigned the same sentiment, while the words with opposite meanings will be assigned the opposite sentiment (Birjali, Kasri & Beni-Hssane, 2021). In a corpus-based approach, sentiment is assigned based on the co-occurrence and syntactic patterns (Birjali, Kasri & Beni-Hssane, 2021).

2.2.2 Machine Learning Methods

Machine learning methods classify documents by identifying patterns and other features in the data that indicate a particular sentiment (Kannan et al. 2016). Commonly the *bag-of-words* model is used to break down the document into smaller groups of consecutive words (*n-grams*) which are then used as input to machine learning algorithms (Kannan et al. 2016). Machine learning methods can be classified in three subgroups: *supervised, unsupervised, semi-supervised*.

Supervised methods are trained on the large, labeled dataset using linear (e.g., Support Vector Machine), probabilistic (e.g., Naïve Bayes), or rule-based methods, or decision trees (Birjali, Kasri and Beni-Hssane, 2021). Unsupervised methods are used when training data don't have

specified labels. In sentiment analysis, hierarchical clustering, and partition methods (K-Means) are commonly used for grouping data based on their similarity (Birjali, Kasri and Beni-Hssane, 2021). Semi-supervised methods use both labeled and unlabeled data to train the model. The advantage of these methods is that they omit the need to collect large amounts of labeled data, but still benefit from the information gained by the supervision (Chapelle, Schölkopf & Zien, 2010). Semi-supervised approaches used in sentiment analysis are generative, co-training, self-training, graph-based and multi-view learning (Birjali, Kasri & Beni-Hssane, 2021).

2.3 Deep Learning Methods

Deep learning methods have gained a lot of attention over the past two decades and are today used extensively for solving different NLP tasks. An advantage of deep learning over traditional methods is their ability to capture long-distance dependencies between words and their relative sentiment (a sentiment which "changes depending on the context" (Joshi, Bhattacharyya & Ahire, 2017, p.89)).

This section provides a brief summary of the neural network architectures used for sentiment analysis.

2.3.1 Word Embeddings

Solving text classification problems, such as sentiment recognition, requires training complex models on a large corpus, which is computationally very expensive. As the interest in text classification has grown, *transfer learning* has been seen as a solution for more efficient training. The idea behind transfer learning is that test and training data don't have to come from the same distribution, and therefore a model that is trained for one task can be used as an input to a model trained for another task (Tan, Sun, Kong, Zhang, Yang & Liu, 2018). In the NLP field, this idea has become very popular for generating word embeddings and several pre-trained models have been developed.

Word embeddings are used to represent words as vectors in the d-dimensional feature space (Zhang, Wang & Liu, 2018). The first pre-trained word embedding model was introduced by Bengio, Ducharme and Vincent (2000). It is a probabilistic language model which generates word embeddings by learning "distributed representation for each word and the likelihood function for word sequences at the same time" (Zhang, Wang & Liu, 2018). After that, Collobert and Weston (2008) have proposed a pre-trained model which has a convolutional neural network architecture that can generate word embeddings using little prior knowledge.

In 2013, Mikolov, Chen, Corrado and Dean developed *Word2Vec*, which has become very popular in sentiment analysis. The model has two architectures that can be used for pre-training: continuous bag-of-words (COBW), which predicts the current word based on the neighboring words, and the Skip-gram architecture, which predicts neighboring words given the current word (Mikolov et al., 2013). Another widely used model is *GloVe* (Global Vectors for Word

Representation), a log-bilinear model which computes word vectors based on their co-occurrence in the text. The objective of the model is to learn word vectors such that their dot product approximates a log ratio of the words' co-occurrence probability (Pennington, Socher & Manning, 2014).

The limitation of the aforementioned models is that the learned word representations capture the meaning of the words only in a single context. In 2018, Peters et al. introduced ELMo (Embeddings from Language Models), a pre-trained model which captures deep contextualized word representations from the entire input sequence. The ELMo architecture contains two bidirectional long short-term memory layers. Each layer has a forward and a backward layer which is trained separately. Therefore, it is often referred to as a *shallowly bidirectional* model. Intermediate embeddings obtained from the first layer are passed onto the second layer. The final word embeddings are computed as a weighted sum of input word vectors, intermediate embeddings obtained from the first layer, and intermediate embeddings obtained from the second layer (Figure 7Figure 1). ELMo produces task-specific embeddings, meaning that a separate model has to be used for each NLP task.

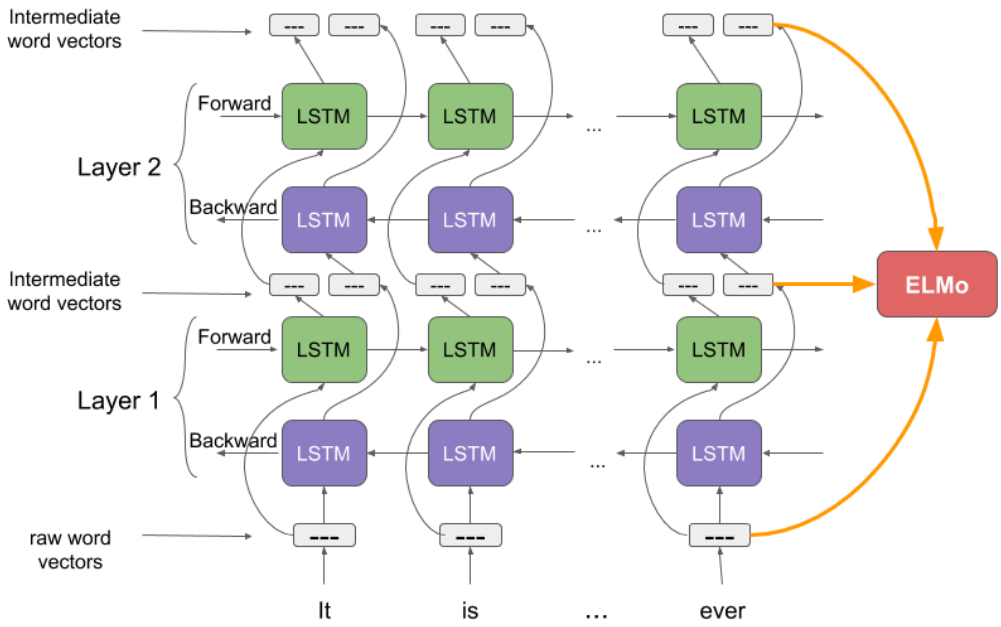

Figure 1 ELMo pre-trained model (Tiwari, 2019)

Besides ELMo, two more models have been developed to generate task-agnostic word representations: GPT and BERT. GPT (Generative Pre-trained Transformer model) was introduced in 2018 by OpenAI (Radford, Narasimhan, Salimans & Sutskever, 2018). The model uses a Transformer decoder structure and has been trained to learn word representations from "left-to-right". BERT (Bidirectional Encoder Representations from Transformers), developed by Google, is a bidirectional language model which uses the transformer encoder structure to learn deep contextual word representations (Devlin, Chang, Lee & Toutanova, 2018).

2.3.2 Recurrent Neural Networks

A recurrent neural network (RNN) is designed to solve tasks that require the handling of sequential data. An RNN consists of a sequence of *hidden states*, where each hidden state at a given step t is computed using the output of the previous hidden state and the input at that step (Figure 2). An important feature of the RNN design is parameter sharing, which has several benefits. Firstly, it allows the network to learn patterns connected to the relative position of words, rather than the absolute position. For example, if we have the sentences "John was at the cinema yesterday" and "Yesterday John was at the cinema", and we want to know when John was at the cinema, we want the model to recognize that it was yesterday regardless of the position of word *yesterday* in the sentence (Goodfellow, Bengio & Courville, 2016). Secondly, it enables the RNN to take an input sequence of any length without changing the number of parameters the network has to learn (Goodfellow, Bengio & Courville, 2016).

The hidden state at each step $h^{(t)}$ is computed based on the previous hidden state $h^{(t-1)}$and the input at that step $x^{(t))}$:

$$h^{(t)} = \sigma(W^{(hh)}h^{(t-1)} + W^{(hx)}x^{(t)} + b^{(t)})$$

where $W^{(hh)}$ and $W^{(hx)}$ are weight matrices and $b^{(t)}$ denote the bias. The output of a hidden state at time t:

$$y^{(t)} = softmax(W^{(hq)}h^{(t)} + b^{(q)})$$

where $W^{(hq)}$ and $b^{(q)}$ denote weight matrix and bias, respectively.

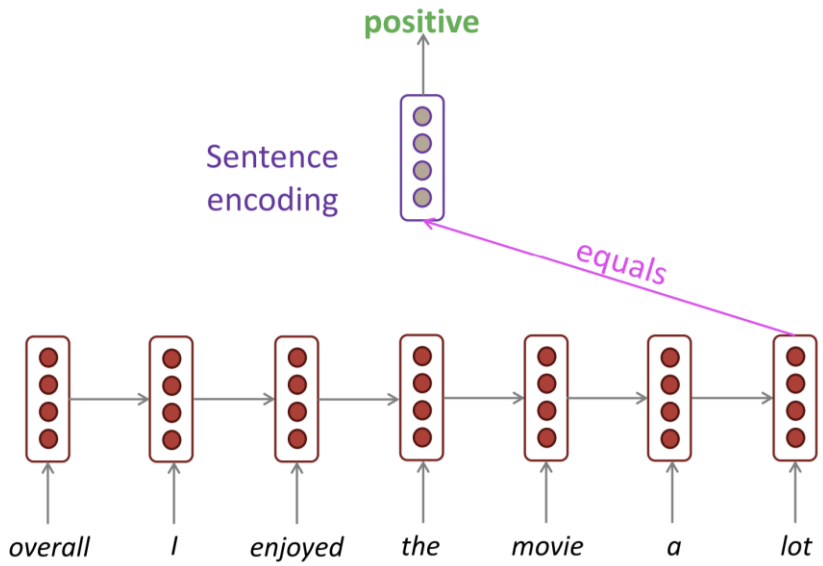

Figure 2 Recurrent neural network for sentiment analysis (Manning, 2022)

RNNs can have many different design patterns, depending on the task they are designed to solve. Karpathy (2015) describes one-to-one, one-to-many, many-to-one and many-to-many

RNN architectures. In the NLP field, a one-to-many architecture is used for generating text from images, many-to-one for sentiment classification, and many-to-many in Name Entity Recognition and Machine Translation (Karpathy, 2015).

The introduction of RNNs marked a substantial improvement over *n-gram* language models and *word-based neural linguistic* models that were used at the time. However, there are a few drawbacks to using RNN. Mohammadi, Mundra, Socher, Wang and Kamath (2019) explain that RNNs suffer from the problem of vanishing and exploding gradients which makes it hard for networks to learn the long-term effects. Another problem he mentions is the long training time, since due to its sequential nature, computation can't be parallelized.

2.3.3 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a special type of recurrent neural network which was designed to mitigate the vanishing gradient problem. This is done by adding a *memory cell,* which is the same dimension as the hidden state, and is used to store long-term information (Zhang, Lipton, Li & Smola, 2021). What will be written and erased from the memory cell is controlled by three gates: *input*, *output* and *forget*. The gates are computed by passing the data through three fully connected layers that use a $sigmoid$ activation function, producing the output that is in the range (0,1) (Zhang et al., 2021).

At time step t, a hidden state $h^{(t)}$ and memory cell $c^{(t)}$ is computed using follow steps:

1. LSTM uses the forget gate layer $f^{(t)}$ to determine which information from the previous memory cell will be remembered and which will be forgotten (Olah, 2015). Using the previous time step hidden state $h^{(t-1)}$, and input at the current time step $x^{(t)}$:

$$f^{(t)} = \sigma(W^{(xf)}x^{(t)} + W^{hf}h^{(t-1)} + b_f$$

2. The input gate layer $i^{(t)}$ decides what new content will be remembered, and computes *candidate memory cell* $\tilde{c}^{(t)}$ which can have values between -1 and 1 (Olah, 2015; Zhang et.al, 2021):

$$i^{(t)} = \sigma(W^{(xi)}x^{(t)} + W^{hi}h^{(t-1)} + b_i$$

$$\tilde{c}^{(t)} = tanh(W^{(xc)}x^{(t)} + W^{hc}h^{(t-1)} + b_c$$

3. The new memory cell $c^{(t)}$ is computed as a sum of element-wise multiplication of the old memory cell by the forget gate (to remove information that should be forgotten), and element-wise multiplication of the candidate memory cell by the input gate (to add information that should be remembered) (Olah, 2015):

$$c^{(t)} = f^{(t)} \circ x^{(t)} + i^{(t)} \circ \tilde{c}^{(t)}$$

4. The hidden state $h^{(t)}$ is computed as the element-wise product of the output gate $o^{(t)}$ and the new memory cell passed through a $tanh$ function (Olah, 2015):

$$o^{(t)} = \sigma(W^{(xo)}x^{(t)} + W^{ho}h^{(t-1)} + b_o$$

$$h^{(t)} = o^{(t)} \circ \tanh(c^{(t)})$$

$W^{(xi)}, W^{(xf)}, W^{(xo)}, W^{(hi)}, W^{(hf)}, W^{(ho)}, W^{(xc)}, W^{(hc)}$ are weight parameters, and b_i, b_f, b_o, b_c are bias parameters.

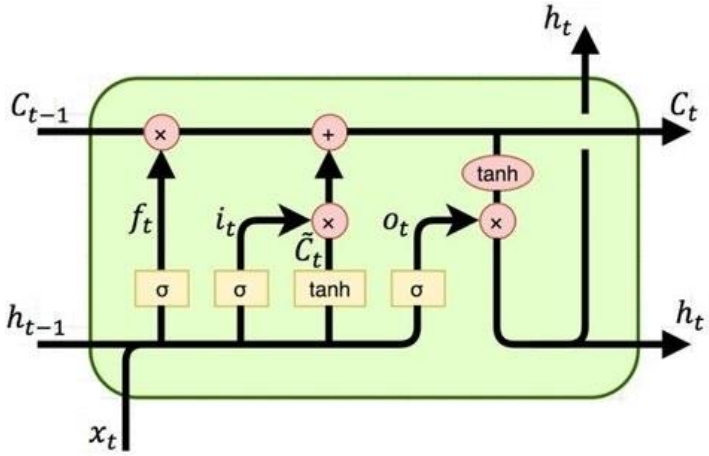

Figure 3 Structure of the LSTM cell (Varsamopoulos, Bertels & Almudever, 2018)

Since LSTM was introduced by Hochreiter and Schmidhuber in 1997, several modifications of the original architecture have been proposed. Gers and Schmidhuber (2000) have modified the original architecture by adding a "peephole connection" between the previous memory cell $c^{(t-1)}$ and the gates, so that the gates depend not only on the previous hidden state but also on the previous cell state (Olah, 2015).

It is important to note that even with LSTMs, the vanishing gradient problem can still occur (Manning, 2021a). However, LSTM has proven to outperform standard RNNs and has become the preferred approach for many NLP tasks (Manning, 2021a).

2.3.4 Bidirectional Recurrent Neural Networks

One limitation of the standard RNN structure is that "[hidden] state at the time t captures only information from the past $x^{(1)}, \ldots, x^{(t-1)}$, and the present input $x^{(t)}$" (Goodfellow, Bengio & Courville, 2016, p.383), meaning that learned "contextual representations only contain information about the left context" (Manning, 2021a, p.51). However, in tasks such as sentiment analysis it is important to learn contextual representations based on the whole sequence. For example, in the sentence "the movie was terribly exciting" the word *terribly* based on the left context will have negative sentiment, which is incorrect (Manning, 2021a).

Bidirectional RNN (Bi-RNN) was designed by Schuster and Paliwal (1997) to address this issue. Bi-RNN contains two layers, a forward and a backward layer, which process the sequence of data simultaneously (Schuster & Paliwal, 1997). The output is generated by concatenating the output of these two layers (Figure 4). A network can have any recurrent architecture, such as traditional RNN, LSTM etc.

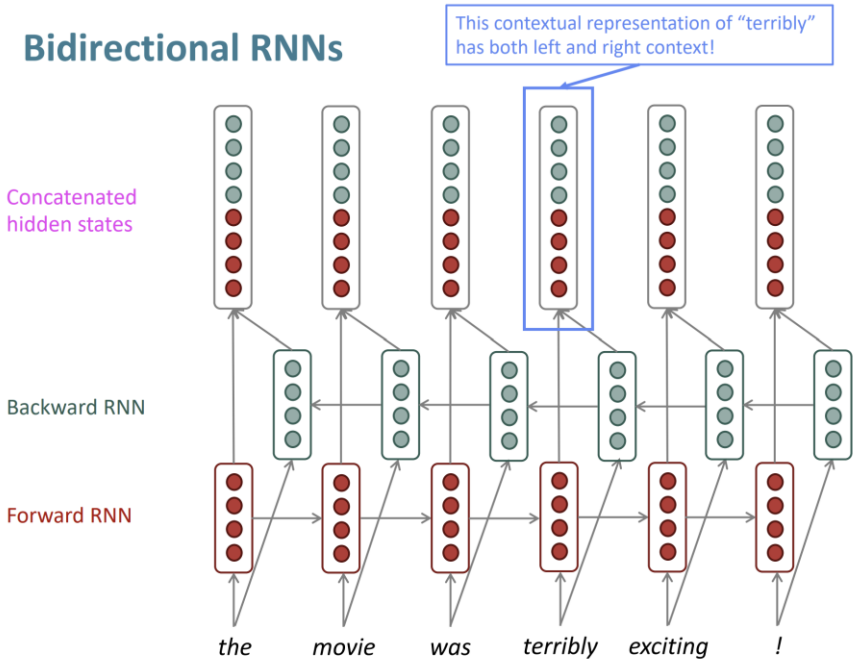

Figure 4 Architecture of bidirectional RNN (Manning, 2021a)

At the time step t, the hidden state of the forward network $\vec{h}^{(t)}$, and the hidden state of the backward network $\overleftarrow{h}^{(t)}$, and the concatenated hidden state $h^{(t)}$ are computed (Manning, 2021a):

$$\vec{h}^{(t)} = RNN_{FW}(\vec{h}^{(t-1)}, x^{(t)})$$

$$\overleftarrow{h}^{(t)} = RNN_{FW}(\overleftarrow{h}^{(t-1)}, x^{(t)})$$

$$h^{(t)} = [\vec{h}^{(t)}; \overleftarrow{h}^{(t)}]$$

where $h^{(t)}$ is the output of the bidirectional network.

2.3.5 Convolutional Neural Networks

Convolutional neural networks (CNNs) were originally used for solving tasks in the field of computer vision. With the development of faster GPUs, researchers developed deeper CNN models, which can extract more complex features. By contrast, the recurrent neural network models could not benefit the same way from this additional computing power. Since in RNNs each time step depends on the previous, computation cannot be parallelized, and networks take a long time to train. Consequently, RNN models are usually very shallow (commonly two

layers) compared to CNN models. The rationale for using deep CNN models for some NLP tasks is that they can extract complex features from the text with the benefit of shorter training time.

Kalchbrenner, Grefenstette and Blunsom (2014) argue that in sentence modeling tasks (e.g., sentiment analysis, machine translation, etc.) word representations should be learned with respect to neighboring words, because identical sentences don't often occur. They have shown that convolutional networks can achieve high performance in sentiment classification tasks. Conneau, Schwenk, Barrault and Lecun (2017) have proposed a deep convolutional network that operates on a character level, which has also achieved good results on the text classification tasks.

Convolution is a mathematical function which measures overlap between two functions, as one function slides over the other (Zhang et al., 2021). In NLP, one-dimensional discrete convolution is used:

$$s(t) = (x * w)(t) = \sum_{a} x(a)w(t-a)$$

where the function of x is input, the function of w is *filter (kernel),* and output is *feature map* (Goodfellow, Bengio & Courville, 2016).

The dimension of the *filter matrix* is equal to the length of the n-gram multiplied by the length of the word embedding vector. Columns in the matrix are called *input channels*. If we have the sentence "tentative deal reached to keep government open", the n-gram size 3, and the word embedding dimension 4, the kernel will slide over the input, multiplying the weights and summing the multiplied values into a single number, as shown in Figure 5.

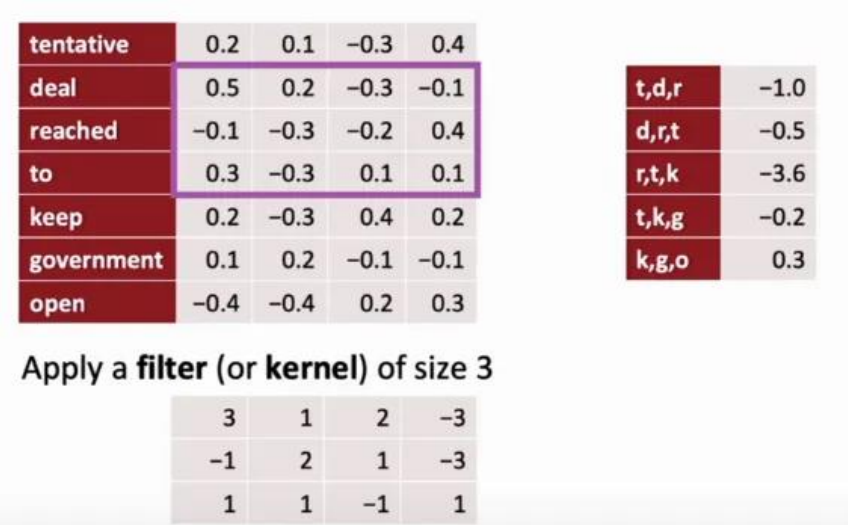

Figure 5 Example of convolution: input (left), output (right), and filter dimensions 3x4 (Manning, 2019)

In Figure 6 we can see that the length of the input doesn't match that of the output of convolution. This can be fixed using *padding*. Padding is used to add a set of zeros to the beginning and the end of the input sentence. To extract more features, it is common to apply multiple filters.

The output matrix provides the features for individual words within a sentence. To obtain features for the whole sentence, the output matrix is summarized using pooling. Furthermore, to produce output that is shorter than the input, the filter's step size, called *stride*, can be increased.

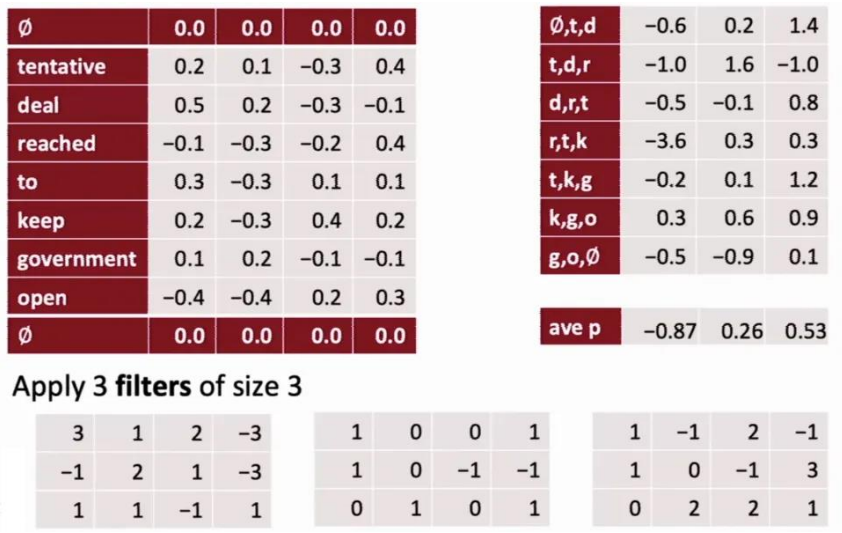

Figure 6 Example of convolution: input (left), output (right). Number of filters is 3, with padding and average pooling through time (Manning, 2019)

2.3.6 Attention

In 2014, Cho, van Merri, Gulcehre, Bahdanau, Bougares, Schwenk and Bengio proposed an architecture for solving *Neural Machine Translation* (NMT) tasks, which has two recurrent neural networks in the encoder–decoder framework (Figure 7). The same year, Sutskever, Vinyals and Le proposed a similar architecture that was based on the LSTM network. In the proposed models, the encoder generates a fixed-length vector representation of the input sequence, which is passed to the decoder (Cho et al., 2014; Sutskever, Vinyals & Le, 2014). The decoder then predicts the output sequence conditioned on the encoder output (Cho et al., 2014).

Although this approach represented a significant advancement in the *Machine translation* field, it still has certain constraints. The issue is that the last hidden state of the encoder

> "needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences..." (Bahdanau, Cho & Bengio, 2014, p.1).

This problem is often referred to as an *information bottleneck*. Bahdanau, Cho and Bengio (2014) have proposed the *attention mechanism* as a solution to this problem. Instead of processing the entire sequence at once, the attention mechanism breaks down the input sequence into smaller segments and uses learned weights to determine the importance of each segment at each step of the model's processing. In other words, instead of having a one context vector c that stores information about the whole sequence, "here the probability is conditioned on a distinct context vector c_i for each target word y_i" (Bahdanau, Cho & Bengio, 2014, pp.3). The target word probability conditioned on the input word is defined as:

$$p(y_i|y_1, \dots, y_{i-1}, \boldsymbol{x}) = g(y_{i-1}, s_i, c_i)$$

where y_{i-1} is the target word of the previous hidden state, s_i and c_i are RNN decoder hidden state and context vector at the time i, respectively.

Let $h_1, \dots, h_n \in \mathbb{R}^h$ denote encoder hidden states, and $s_t \in \mathbb{R}^h$ the decoder hidden state in the time step *t*. The attention computes how important different parts of the input sequence are for generating the target word y_t:

- First, the attention scores e^t are calculated as a dot product of the encoder hidden states and the decoder's hidden state at the time step t:
$$e^t = [s_t^T h_1, \dots, s_t^T h_n] \in \mathbb{R}^N$$
- Then, the attention distribution at the time step t is calculated using a $softmax$ function:
$$\alpha^t = softmax(e^t)$$
- The output of the attention called *context vector* is a weighted sum of the encoder hidden states:
$$a^t = \sum_{i=1}^{N} \alpha_i^t \, h_i \in \mathbb{R}^h$$
- Last, the attention output a^t and decoder hidden state s^t are concatenated $[a^t; s^t] \in \mathbb{R}^h$ and squashed through a $softmax$ function to obtain the probability distribution of output words.

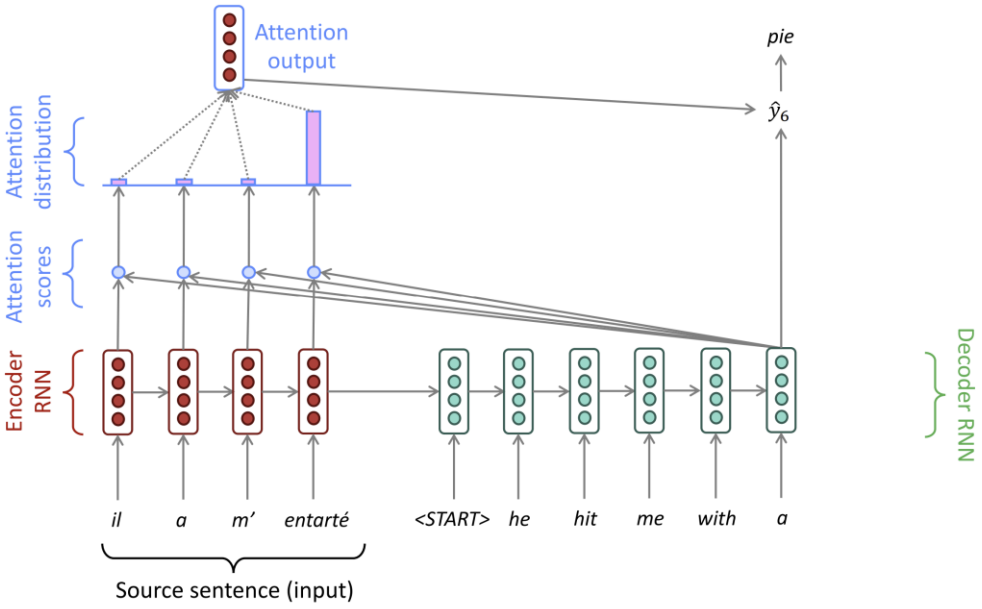

Figure 7 Sequence-to-sequence with attention (Manning, 2021b)

2.4 Transformer

Transformer is a neural network architecture designed to solve sequence-to-sequence tasks. It uses the attention mechanism to capture complex relationships between words and to learn different meanings that words have based on their given context. The main idea behind the Transformer is that significant computational efficiency can be achieved by replacing the RNN layers in the architecture described in section 2.3.6 with attention layers. This omits the need for sequential processing and significantly improves training time.

In 2017, Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser & Polosukhin introduced the Transformer in the article *Attention Is All You Need*. The model consists of an encoder and a decoder. The encoder takes word embeddings together with positional encodings as the input and generates updated word embeddings which have a better contextual understanding of the input words. These updated word embeddings are then used by the decoder to generate an output sequence.

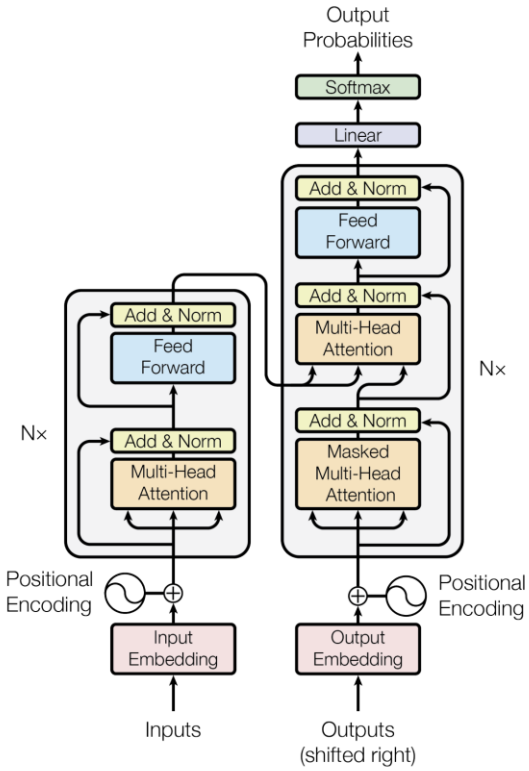

Figure 8 Transformer model architecture (Vaswani et al., 2017)

The encoder part is made from six stacked layers, where each layer has two sublayers: a *multi-head self-attention mechanism* and a *position-wise fully connected layer* (Vaswani et al., 2017). Let's suppose that we are training a model that translates text from English to German. The input to the encoder is word embeddings and positional encodings of all words in the English sentence. The multi-head self-attention sublayer takes the input and computes the attention for each word, which is then passed to the feed-forward sublayer. This sublayer computes new word embeddings for English words, which capture complex contextual relationships between words. In order to improve the training, residual connections are implemented around both sublayers, and, subsequently, layer normalization is applied (Vaswani et al., 2017).

In the training phase, the input to the decoder is word embeddings and position encoding of all words in the German sentence. The input first goes through the masked multi-head attention sublayer which computes the attention for each German word. Then, in the second sublayer, multi-head cross attention establishes a connection between English and German words. After that, the output of the second sublayer is forwarded to the positional feed forward network. Like in the encoder, residual connections are implemented around both sublayers, and, subsequently, layer normalization is applied.

Finally, the output of the decoder is resized to the size of the German dictionary, and the probability is computed for each word.

2.4.1 Encoder

The role of the encoder is to produce contextual word embeddings for the input sequence. As mentioned, the encoder consists of two sublayers: a multi-head self-attention mechanism and a position-wise fully connected layer. This section explains all the elements of the encoder and their roles.

Self-attention (Scaled Dot-Product attention)

The difference between the attention mechanism described in section 2.3.6 and self-attention is that "instead of relating an input to an output sequence, self-attention focuses on a single sequence" (van Dongen, 2023). Each word in a sequence is compared to every other word in the sequence, and the attention distribution is determined based on these comparisons.

Vaswani et al. (2017) describe the attention function as "mapping a query and a set of key-value pairs to an output where the query, keys and values, and output are vectors" (p.3).

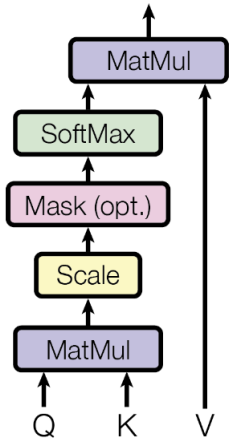

Figure 9 Self-attention (Scaled Dot-Product attention) (Vaswani et al., 2017)

Let $x_1, x_2, \ldots, x_t$ denote embeddings for t input tokens, where $x_i \in \mathbb{R}^d$:

- value $v_i = W^v x_i$, where $W^v \in \mathbb{R}^{d_v \, x \, d}$ is the value matrix and $v_i \in \mathbb{R}^{d_v}$,
- key $k_i = W^k x_i$, where $W^k \in \mathbb{R}^{d_k \, x \, d}$ is the key matrix and $k_i \in \mathbb{R}^{d_k}$,
- query $q_i = W^q x_i$, where $W^q \in \mathbb{R}^{d_k \, x \, d}$ is the query matrix and $q_i \in \mathbb{R}^{d_k}$.

W_v, W_k and W_q are the weight matrices which are learned during the training (Raschka, 2023).

Since attention is computed for a set of queries at the same time, values, keys, and queries can be written as $V = XW^v,\ K = XW^k,\ Q = XW^q$ (Goldie & Hewitt, 2022a).

To compute self-attention:

- First the attention scores matrix is calculated by performing matrix multiplication of the key and query matrices. The attention scores are then scaled by $\sqrt{d_k}$. Vaswani et al. (2017) explain that the reason for scaling is the gradient stability, as "for large values of

d_k, the dot products grow large in magnitude, pushing the $softmax$ function into regions where it has extremely small gradients" (p.4).

- Attention scores are normalized using a $softmax$ function, and the weighted sum is calculated using matrix multiplication:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Head Attention

Instead of single attention mechanism, the multi-head attention computes multiple attentions called *attention heads*. Each attention head focuses on different aspects of the input data.

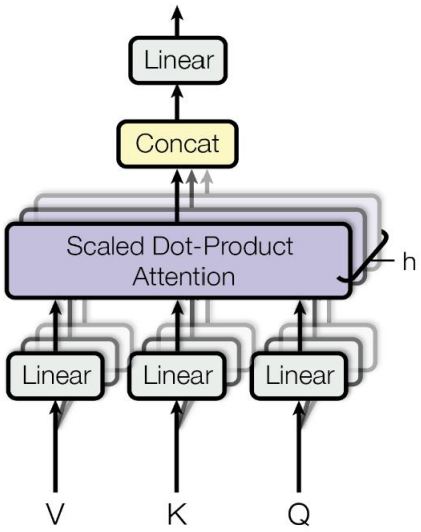

Figure 10 Multi-head attention (Vaswani et al., 2017)

Instead of a single Q, V, K matrix, in the multi-head attention there is l number of matrices Q_l, V_l, K_l dimension $\mathbb{R}^{d_k \, x \frac{d}{h}}$, where $l = 1, \ldots, h$, and h is the total number of attention heads (Goldie & Hewitt, 2022a). Each attention head learns its own set of matrices Q_l, V_l, K_l independent from other heads. The output vectors of the heads are then concatenated into a single $\mathbb{R}^{d \, x \, d}$ matrix, and the result of the multi-head attention is a linear transformation of this matrix. Transformer architecture suggested by Vaswani et al. (2017) have 8 heads, however this is a hyperparameter that is tuned during training (Alammar, 2018; Doshi, K., 2021).

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^o$$
$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

The good results achieved by the transformers can be primarily attributed to the multi-head attention (Liu, Liu & Han, 2021). It allows the model to attend to different positions in the input sequence, which is very beneficial when dealing with long and complex relationships between

words (Hewitt, 2021). Figure 11 shows an example of the multi-head attention mechanism where heads are capturing the long-distance dependencies between the word *making* and the words *more* and *difficult*.

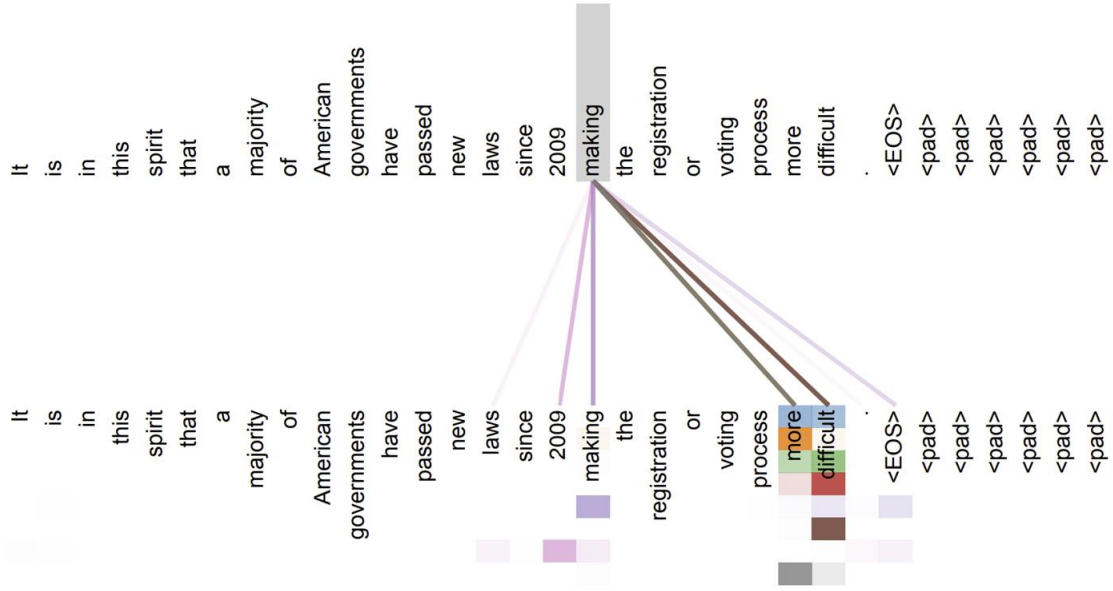

Figure 11 An example of multi-head attention mechanism. Each attention head is represented by a different color (Vaswani et al., 2017)

Residual Connections

The residual connection framework was proposed by He, Zhang, Ren and Sun (2016) as a better way of training a deep network that suffers from the vanishing gradient problem and degradation of accuracy. In the residual connection framework, *shortcut connections* are used to add the output of one layer to the input of a later layer in the deep network. This improves information flow through the network and enables the gradient to propagate more easily (Hewitt, 2021).

Let $H(x)$ be the underlying mapping to be fit by a few stacked layers $F(x)$, and x be the input to the first of these layers. Then $F(x) = H(x) - x$, and therefore $H(x) = F(x) + x$.

The hypothesis of He et al. (2016) is that it is easier to optimize the residual mapping than the underlying mapping. In other words, instead of learning the $H(x)$ from scratch, the model will only learn the difference between the previous and the current residual block.

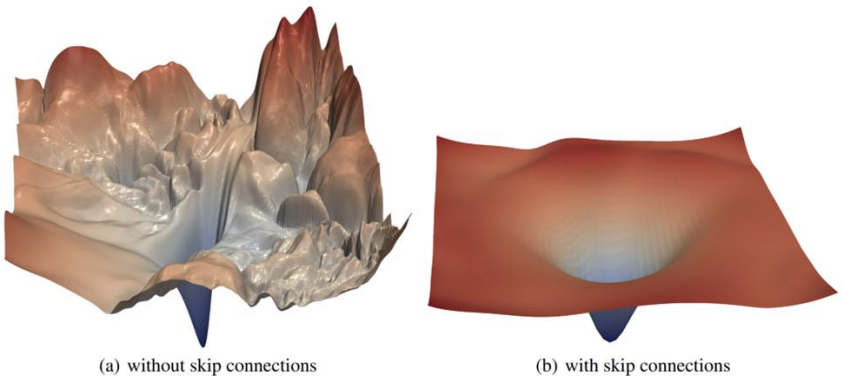

Figure 12 Surface of the loss function without (left) and with (right) "shortcut connections" (Li, Xu, Taylor, Studer & Goldstein, 2018)

Layer Normalization

During the training, gradient values are highly impacted by the output of the previous layer (Ba, Kiros & Hinton, 2016). Normalization reduces this impact by removing uninformative variation, which as a result, smooths the loss function and speeds up the model training. If $x_1, \dots x_n$ where $x_i \in \mathbb{R}^d$, layer normalization for each input x_i computes separate parameters $\hat{\mu}_i$ and $\hat{\sigma}_i$ across all *d*-dimensions.

In the Transformer model, layer normalization $LayerNorm(x + Sublayer(x))$ is applied after each sublayer (Vaswani et al., 2017). Let $h_1, \dots h_n$ where $h_i \in \mathbb{R}^d$ denote contextual representations of the input computed by the attention mechanism. The normalization parameters are calculated for each representation h_i separately:

$$\hat{\mu}_i = \frac{1}{d}\sum_{i=1}^{d} h_{ij}$$

$$\hat{\sigma}_i = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(h_{ij} - \hat{\mu}_i)^2}$$

where $\hat{\mu}_i$ and $\hat{\sigma}_i$ are scalars. To compute the $LayerNorm$ $\hat{\mu}_i$ and $\hat{\sigma}_i$ are replicated across all dimensions of h_i:

$$LayerNorm(h_i) = \frac{h_i - \hat{\mu}_i}{\hat{\sigma}_i}$$

Position-Wise Feed Forward Network

In both the encoder and decoder, attention layers are followed by a fully connected feed forward network. Hewitt (2021) explains that since the attention is computing the weighted sums, non-linearity is needed for the model to learn complex relationships:

$$FFN(x) = \max(0, xW_1 + b_1)\, W_2 + b_2$$

where W_1, W_2 are weights and b_1, b_2 bias terms of the first and second layers, respectively. The network consists of two linear transformations with a $ReLU$ activation function in between. The first linear transformation projects the input into the higher dimensional space ($d_{ff} = 2048$), and the second one projects it back into original dimension ($d = 512$).

Positional Encoding

Self-attention does not consider the position of the word in the sentence. The Transformer model addresses this issue by adding *positional encoding* to the input embedding of the encoder and decoder block.

Let *pos* denote the position embedding, which is the same dimension as the embedding vector d_{model}, and i denotes indices of each position embedding dimension. Encoding of words in odd positions in the sequence is done using a $sine$ function, and in even positions using a $cosine$ function:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

The given functions indicate that frequencies decrease as the vector dimensions increase, forming a geometric progression from 2π to $10000 * 2\pi$ on the sinusoid's wavelengths.

2.4.2 Decoder

The decoder consists of three sublayers: *masked multi-head self-attention*, *multi-head cross attention* and *feed-forward network*.

During the training phase, input to the decoder are word embeddings and position encodings of all words in the target sequence. In the previous example of a model that translates text from English to German, the input is the sequence in German.

The first sublayer performs *masked multi-head self-attention*. In the decoder, *masking* is used to prevent a word from attending to words that come after it in the sequence. This stops the model from "looking into the future" and enables it to learn to predict words based on the left-hand side content (Vaswani et al., 2017). During the training, a masking layer assigns an unnormalized attention score of $-\infty$ to all words in the sequence following the word that the attention is computed for (Figure 13). Once squashed through a $softmax$ function, the normalized attention scores for these words will be equal to zero.

	Martha	hugged	her	son
Martha		-∞	-∞	-∞
hugged			-∞	-∞
her				-∞
son				

Figure 13 Masking in self-attention. Words in each row have words in the right-hand side masked out (e.g., "hugged" can only attend to "Martha" and "hugged").

The second sublayer, *multi-head cross attention*, performs encoder–decoder attention. It takes the output of the first layer of the decoder (query) and attends it to the output of the encoder (keys, values), establishing a connection between the input and the output. In the example of the English–German translation model, during this step, each German word is mapped to the English words that have similar meanings. As a result, the new word embeddings of German words will contain an English context. The third sublayer is the *feed-forward network*. Just as in the encoder, it is used to add non-linearity to the model.

Using the linear transformation, the output of the decoder is transformed to the dimension which is the same size as the dictionary (in the translation model example, this is the dictionary of German words generated from the training dataset). Then, for each word in the sequence, a $softmax$ function computes the probability distribution over the dictionary and returns the word with the highest probability score (the most probable German word).

During the inference phase, instead of the target sequence, input to the decoder at the time step t is the output of the decoder at the previous time step $t-1$. Therefore, each word w_t is generated based on the input from the encoder and the previous $w_1, \dots, w_{t-1}$ words.

2.5 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained masked language model, introduced by Google in 2018. The pre-trained language models developed before BERT were either using the unidirectional approach, where they learned deep contextual representation from left-to-right (such as GPT), or they used the shallow concatenation of a left-to-right and a right-to-left context (such as ELMo). The novelty of BERT is its ability to learn word representations that are jointly conditioned on the left and the right context (Devlin et al., 2018).

The model uses the transformer encoder structure. Furthermore, BERT uses masking to prevent the attention mechanism from seeing all words during training, allowing model to overcome the limitations of the unidirectionality (Devlin et al., 2018).

The BERT implementation has two phases: the *pre-training phase*, during which the model is trained on unlabeled data, and the *fine-tuning phase*, during which model parameters are fine-tuned using the labeled data from a downstream task (e.g., text classification, question answering, etc.) (Devlin et al., 2018). Model pre-training is a language modeling task where, by learning to predict the next word based on the previous words in the sequence, the model derives general patterns from text, building a general understanding of the language. To train the model for the downstream task, hidden layers (optionally) and the output layer are added to the pre-trained model. During the fine-tuning phase, the model learns task-specific parameters, and the parameters of the pre-trained model are slightly adjusted. Alternatively, instead of fine-tuning, *a feature-based* approach can be applied, in which the pre-trained model parameters are frozen and the model learns only task-specific parameters.

2.5.1 Input/Output Representation

BERT can take either a single sequence or a pair of sequences (e.g., question–answer) as input. Words are transformed into tokens using a sub-word type of tokenizer *WordPiece*, developed by Google (Wu, Schuster, Chen, Le, Norouzi, Macherey, Krikun, Cao, Gao, Macherey & Klingner, 2016). The details on how the tokenizer is implemented in BERT are not publicly available. However, the company HuggingFace has published the implementation description based on the publicly available literature that they have gathered. According to HuggingFace (n.d)

- the tokenizer first breaks each word into elements at the letter/sign level and adds the prefix ## before each character except the first one (e.g., the word “book” will be split into $[b, \#\#o, \#\#o, \#\#k]$)
- a score is calculated for each element pair and used to decide whether to concatenate them using the formula

$$score = (freq\_of\_pair)/(freq\_of\_first\_element \times freq\_of\_second\_element)$$

- if individual elements of an element pair are less common in the vocabulary than element pair itself, they will be concatenated.

HuggingFace (n.d.) explains that if the elements “un” and “##able” very often appear in other words, their frequency will be higher than the frequency of the pair [“un”, “##able”], thus they will not be concatenated.

The input sequence into the BERT model always starts with a class token [CLS], which is a sequence representation, and ends with a separating [SEP] token. If the input consists of two sequences (e.g., question–answer), they are separated by a [SEP] token. The input representation is the sum of token embedding, segment embedding (indicating which segment a token belongs to), and positional embedding (indicating the position of a token in the sequence) (Figure 14).

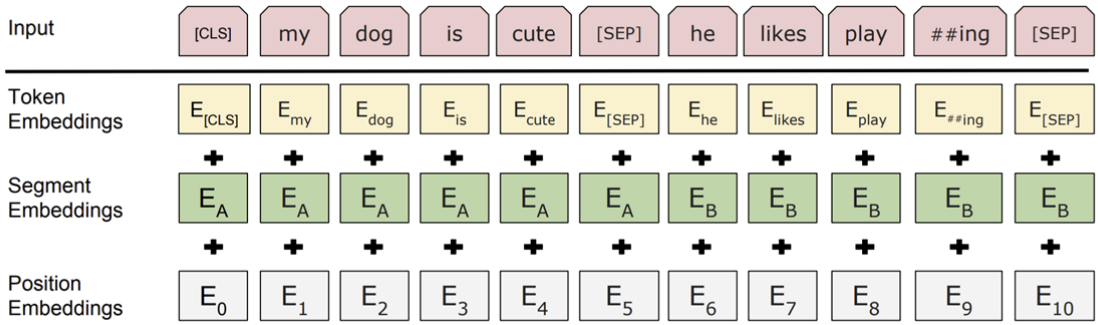

Figure 14 BERT input representation (Devlin et al., 2018)

2.5.2 Pre-Training

BERT was pre-trained on the BooksCorpus (Zhu, Kiros, Zemel, Salakhutdinov, Urtasun, Torralba & Fidler, 2015) and English Wikipedia, which has 800 million and 2,500 million words, respectively. The *base BERT* model consists of 12 layers (transformer encoder blocks), 768 hidden states, and 12 attention heads, resulting in 110 million parameters (Devlin et al.,2018). The *large BERT* model has 24 layers (transformer encoder blocks), 1024 hidden states, and 16 attention heads, resulting in 340 million parameters (Devlin et al., 2018). Pre-training consists of two unsupervised tasks which are conducted simultaneously: *Masked Language Modeling* and *Next Sentence Prediction*.

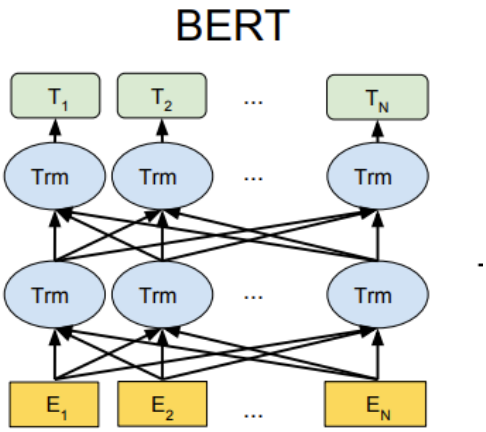

Figure 15 Architecture of the BERT pre-trained model. $E_1 \dots E_n$ are the input to the model, $T_1 \dots T_n$ are output of the model, and T_{rm} are transformer blocks (Devlin et al., 2018)

Masked Language Modeling (MLM)

Devlin et al. (2018) states that "standard conditional language models can only be trained left-to-right or right-to-left, since bidirectional conditioning would allow each word to indirectly *see itself"*. The same problem applies to the transformer encoder, since the attention mechanism has access to the whole sentence (including the word it is trying to predict). To address this issue Devlin et al. (2018) employ masking. The masking was inspired by the *Chloze task* (Taylor, 1593) of filling in the missing words in a text. Rather than predicting the next word given the previous words, the model predicts the missing words given the rest of the sequence.

BERT randomly samples 15% of the tokens to apply masking on. 80% of the sampled data is replaced with the [MASK] token, 10% with a random word, and 10% is unchanged (Devlin et al., 2018). The objective of the model is to predict sampled words. The reason behind this masking scheme is to reduce the mismatch between pre-training and fine-tuning, since the masking is used only during the pre-training phase (Devlin et al., 2018). If the whole sample was replaced with [MASK] token, the model will learn to predict only masked words. Furthermore, it introduces uncertainty to the model since the model doesn't know if the sampled token that is not replaced with a [MASK] token represents the actual word or a random word. In Figure 16, an example is given of a sentence where three words are sampled for prediction: the word *went* is replaced with *pizza*, *to* is the actual word, and *store* is masked. The sequence has one [MASK] token, but it will have three loss terms.

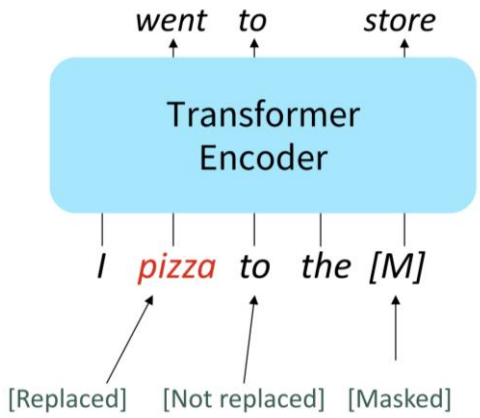

Figure 16 Masking example (Goldie & Hewitt, 2022b)

Next Sentence Prediction (NSP)

The goal of the Next Sentence Prediction is for the model to learn the relationships between two different segments of text, as this is highly important for some NLP tasks such as question answering (Devlin et al., 2018). The model is a binary classifier trained on sequence pairs, where consecutive sequences are labeled *IsNext*, and non-consecutive sequences are labeled *NotNext* (Figure 17) (Devlin et al., 2018).

```
Input = [CLS] the man went to [MASK] store [SEP]
        he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
        penguin [MASK] are flight ##less birds [SEP]
Label = NotNext
```

Figure 17 Example of training dataset used for NSP (Devlin et al., 2018)

2.5.3 Fine-Tuning and Feature-Based Approach

As mentioned, there are two approaches for how BERT can be used for the downstream task: fine-tuning and a feature-based approach. The choice of method depends on the type of task and size of the training dataset.

Fine-tuning is the preferred way of applying BERT to the downstream task, as it allows the model to learn domain-specific knowledge. If the dataset is sufficiently large, fine-tuning will, in most cases, improve the model's performance. During the fine-tuning, backpropagation goes through all layers of the network, updating all weights (including pre-trained weights). In order to avoid overfitting, the general knowledge of the pre-trained model needs to be preserved. Therefore, the weights of the pre-trained model are only slightly changed. The illustration of fine-tuning BERT for the classification task is given in Figure 18.

The feature-based approach is suitable for situations where tasks cannot be modeled by the Transformer encoder architecture, or when the training dataset is very small and fine-tuning will lead to a decrease in the model performance. Another benefit is a lower computational cost compared to fine-tuning. In the feature-based approach, the model updates only the task-specific weights. This is done by freezing the layers of the pre-trained model, which allows backpropagation to go through only the layers added for the downstream task.

Layer freezing is sometimes used in combination with fine-tuning. In this approach, only some layers of the pre-trained model are frozen, while others get updated.

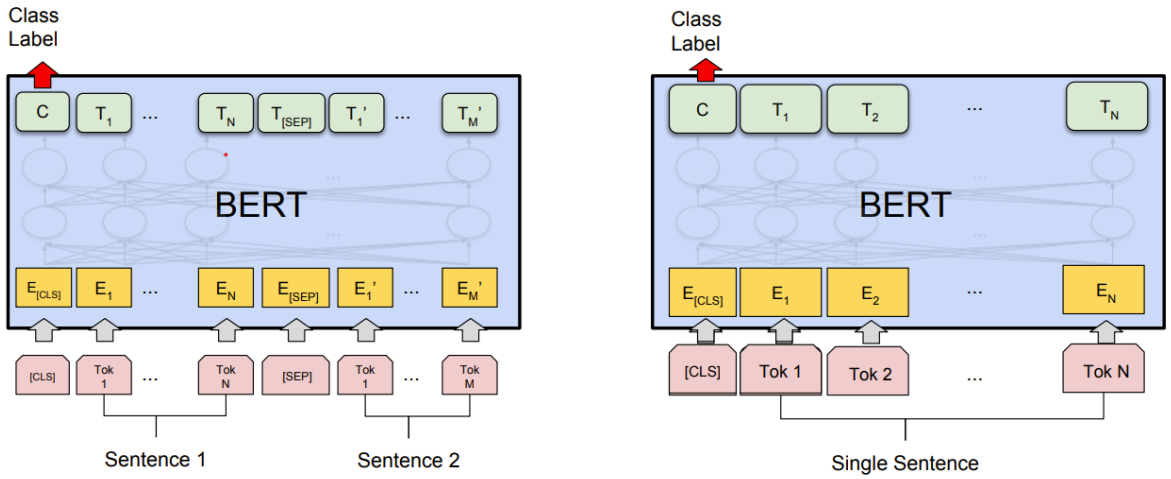

Figure 18 Fine-tuning BERT for a Sequence Pair Classification Task (left), and Single Sentence Classification Task (right) (Devlin et al., 2018)

2.5.4 Model Output

HuggingFace's implementation of the BERT model provides four different outputs that can be used depending on the downstream task. In classification, either the *pooler_output* or the *last_hidden_state* is used. The pooler_output returns the sequence classification token [CLS] from the last layer hidden state, after it has passed through the linear layer and *tanh* activation function (i.e., sequence embedding). The output dimension is [batch size, hidden size]. The last_hidden_state returns the last hidden state's output embeddings of all tokens. The output dimensions are [batch size, sequence length, and hidden size] (Figure 19). To obtain the sequence embeddings, max pooling or average pooling is commonly used. When it comes to the effect on the performance of the classification model, none of the outputs is superior. However, experience from practical implementation shows that in most cases the pooler_output provides better results when the model is fine-tuned, and last_hidden_state when the model weights are frozen.

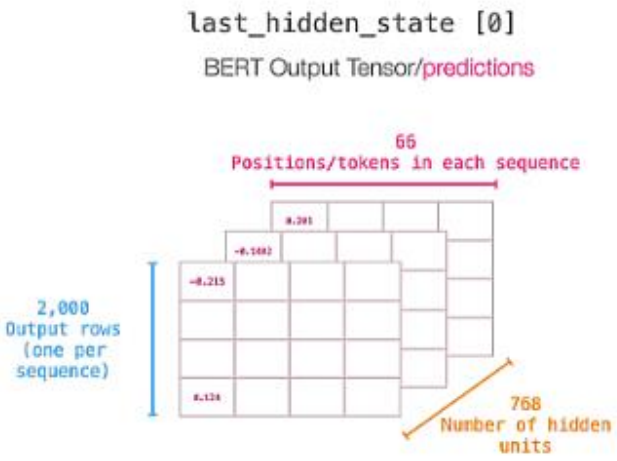

Figure 19 Dimensions of the BERT model last hidden state (Singh, 2017)

3 Data

The dataset (DG, 2021) used in the analysis consists of 838 566 reviews left by the current and former employees of companies from the United Kingdom in the period between January 2008 and June 2021 on the employer review platform Glassdoor. The full list of the dataset variables is given in Table 1. The dataset was downloaded from the Kaggle platform, where it was published under the name "Glassdoor Job Reviews" by the user DG in 2021.

Table 1 Glassdoor Job Review dataset - variables overview (Scale takes values from 1 to 5, where 1 = very dissatisfied, 5 = very satisfied)

Name	Type	Description
firm	categorical	Name of reviewed company
date_review	date	Date of the review
job_title	categorical	Job position of reviewer
current	categorical	Reviewer status. Values: current employee and former employee
location	categorical	Job location
overall_rating	integer	Overall rating is average value of variables work_life_balance, culture_value, diversity_inclusion, career_opp, comp_benefits, senior_mgm, recommend, ceo_approv.Values: from 1 to 5
work_life_balance	integer (scale)	Work-life balance
culture_value	integer (scale)	Company culture
diversity_inclusion	integer (scale)	Diversity and inclusion at the workplace
career_opp	integer (scale)	Opportunity for career development
comp_benefits	integer (scale)	Work benefits
senior_mgm	integer (scale)	Satisfaction with senior management
recommend	integer (scale)	Would the reviewer recommend the company to a friend
ceo_approv	integer (scale)	Opinion of review about CEO
outlook	integer (scale)	Reviewer opinion regarding the future of the company
headline	categorical	Headline of the review
pros	categorical	Positive aspects of working for the company
cons	categorical	Negative aspects of working for the company

3.1 Dataset Exploration

The dataset contains reviews by the employees of 428 companies. Most of the companies in the dataset are rated positively, receiving a score of 4 or 5 (Figure 20). Only 14% of the companies in the dataset have been rated with an overall rating under 3. Furthermore, Figure 21 shows the ratings of twenty companies that have received the most reviews, showing significant variation in the number of reviews among the companies.

Most reviewers chose to be anonymous or left the job title field blank (Figure 22). Additionally, many of the reviews come from employees who work (or have worked) in middle or senior positions in the companies.

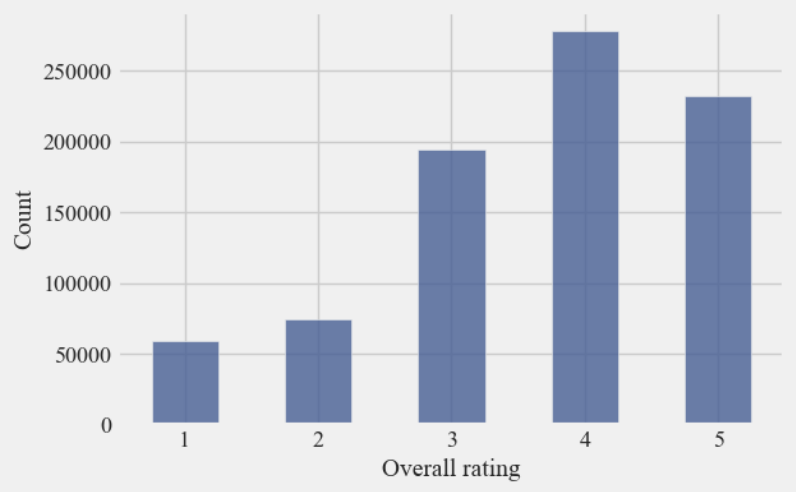

Figure 20 Distribution of overall ratings

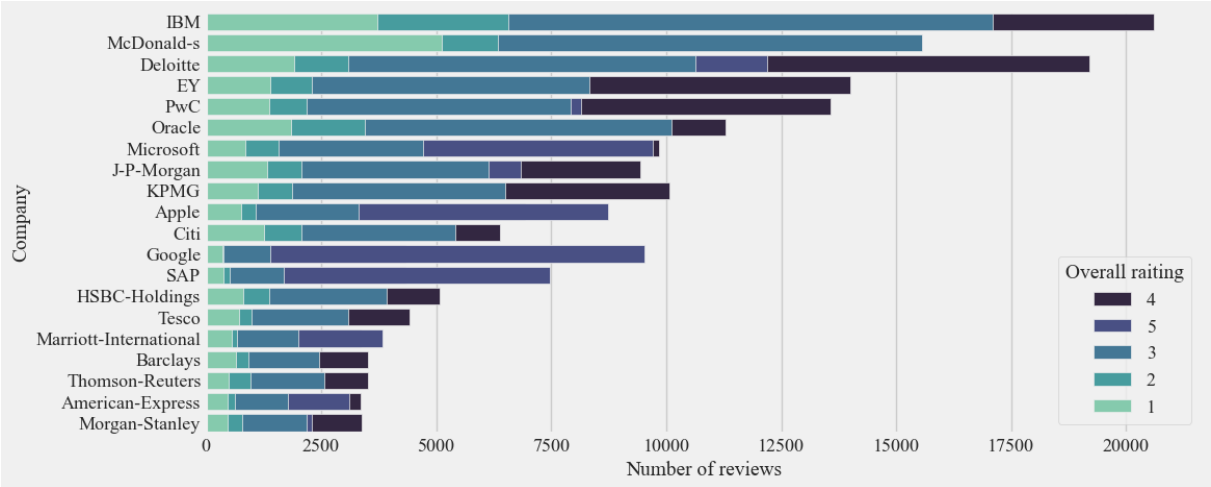

Figure 21 Ratings of twenty companies with largest number of reviews (Color assigned by total count)

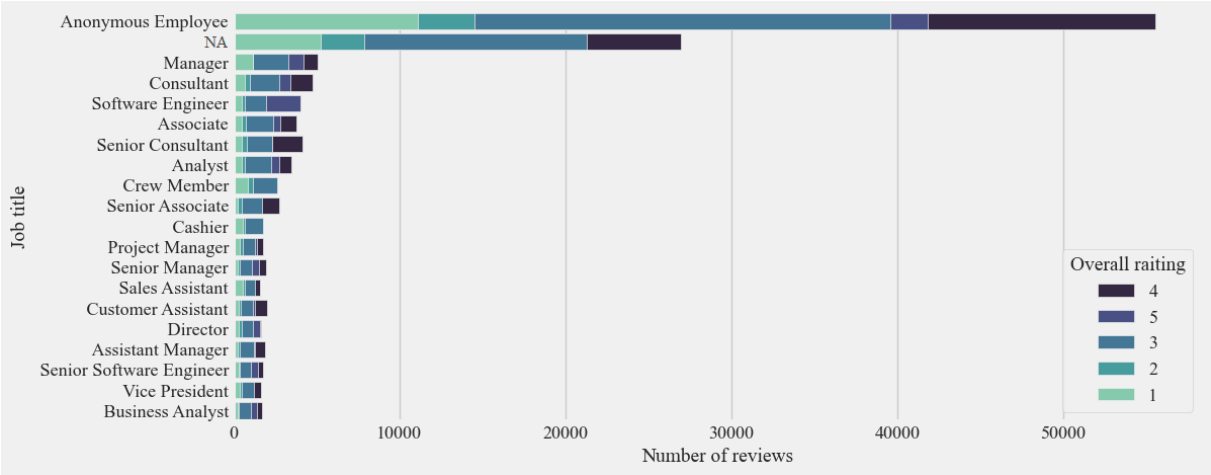

Figure 22 Distribution of rating over twenty most common job titles (Color assigned by total count)

A sample review from each of the "overall rating" categories is provided in Table 2. In this sample we can see that improper use of spacing and double spacing is a common typing mistakes in the text. This information is useful when deciding which data cleaning measures need to be taken.

Table 2 Examples of reviews

Company: BREWIN-DOLPHIN **Job Position:** Analyst **Headline:** If you're into a cut-throat environment, this place is for you. **Pros:** Some of the clients. Free biscuits and booze. **Cons:** 'Management' are cold. The only word to describe them is corrupt. Never expect anything to be said to your face, and never expect them to be on your side - one in particular is the infamous wimp of the office. Such a pity, as I had such loyalty to the company, but I can't see myself lasting much longer in here unless I move offices (and they're probably just as bad as this lot). Awful. **Rating:** 1.0
Company: IBM **Job Position:** Analyst **Headline:** NA **Pros:** Company brand with long history **Cons:** Low salary and following with low salary increment; Hierarchy Culture is an issue; Heavy workload at junior level only ; Too many compliance/rules; Slow decision making ; Many bottleneck in the company and less or no development opportunity ; Insufficient Training **Rating:** 2.0

Company: MORGAN-STANLEY
Job Position: Executive Director
Headline: Uncaring
Pros: MS consistently places the needs of the company above its employees. I get it, companies do this. But MS does so more than any other company I've known.
Cons: Management doesn't value its talent - their view is we'll treat you as we like, if you don't like it leave. Stingy on compensation and benefits.
Rating: 3.0

Company: DELOITTE
Job Position: Consultant
Headline: Deloitte Review
Pros: Good open culture & maintains work life balance
Cons: networking is a must even if u r weak in technilogy that is fine.
Rating: 4.0

Company: GOLDMAN-SACHS
Job Position: Analyst
Headline: Tests , but rewards you
Pros: Challenging, the pay grade is great,
Being my first job, I have no comparisons, but i can vouch for the place, it's great
Cons: Yes, the hours can get long. Work can at times become quite monotonous, but there are always opportunities to expand and diversify your portfolio
Rating: 5.0

Figure 23 shows the length of the text in the “pros” and “cons” across the “overall rating” variable measured in number of words. The graph shows that most of the reviews have up to twenty words. Furthermore. it indicates reviews with lower “overall rating” tend to be shorter than those with higher ratings.

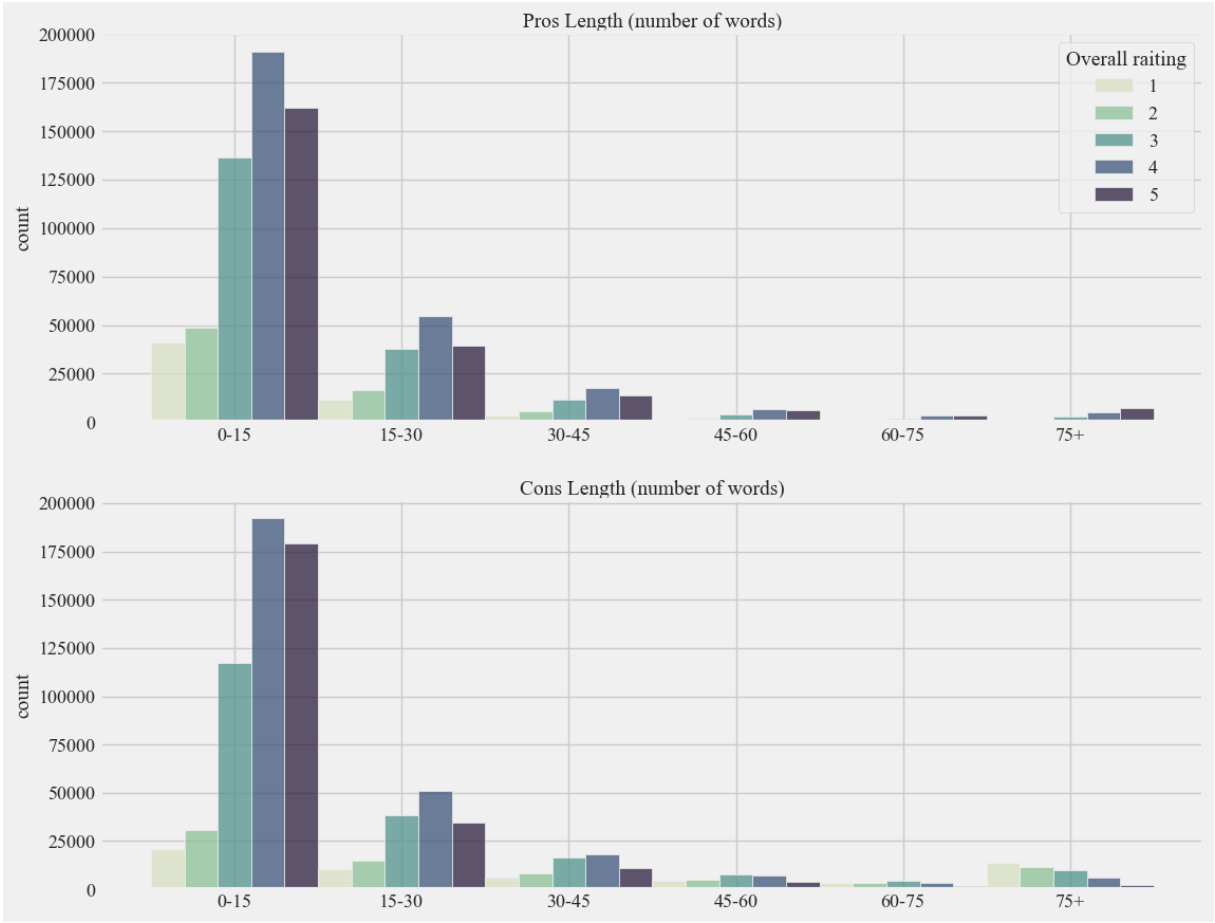

Figure 23 Length of text (number of words) of "pros" (up) and "cons" (down) across overall rating values. Text length is grouped in six ranges

4 Methodology

This chapter describes the details of the research design, model architecture, and model implementation.

4.1 Research Design

To evaluate how the performance of the sentiment classification model is affected by including two additional features, and by the proposed method for doing so, three different models have been trained. The first model classifies the sentiment based only on the review text. The second model concatenates review and additional features prior to tokenization. The third model implements the method proposed in this thesis. The input to the model is the sequence pairs, where the first sequence represents a review, and the second represents the concatenated features “job title” and “firm”.

The first model is used as a benchmark to measure the effect of adding features in the analysis on the sentiment classification results. The second model is used as a benchmark to measure whether the method proposed in this thesis improves the performance of the model.

To ensure the fairness of the comparison with regard to the limitations of the study described in section 5.4, all three models use the same architecture. Furthermore, all models are trained using the same optimizer and the same learning rate.

4.2 Data Preparation

The Glassdoor Job Reviews dataset does not contain sentiment labels. To overcome this, the dataset was converted into a long format by merging the “pros” and “cons” columns into a single column “review”. After the transformation, each observation from the original dataset is represented in two rows, one for “pros” and one for “cons”, resulting in a dataset that is twice the length of the original, containing 1 677 132 rows.

For the purpose of the analysis, the sentiment label needs to be added. This was done using the rule-based approach. In the initial phase of the research, two rulesets were tested. Both rulesets determine the sentiment of the text in the “review” column, using the “overall_rating”, and whether the text originally belonged to the “pros” or the “cons” column. The first ruleset classified the reviews into nine classes and the second one into three classes (Table 3).

Table 3 Rulesets for assigning the sentiment

Ruleset 1 – nine classes
9 - Extremely Positive: text from pros with overall rating is 5
8 - Very Positive: text from pros with overall rating is 4
7 - Moderately Positive: text from pros with overall rating is 3
6 - Slightly Positive: text from pros with overall rating is 2
5 - Neutral: text from pros with overall rating is 1, and cons with overall rating 5
4 - Slightly Negative: text from cons with overall rating 4
3- Moderately Negative: text from cons with overall rating 3
2- Very Negative: text from cons with overall rating 2
1- Extremely Negative: text from cons with overall rating 1
Ruleset 2 – three classes
2 - Positive: text from pros with overall rating is 2, 3, 4, 5
1 - Neutral: text from pros with overall rating is 1, and cons with overall rating 5
0- Negative: text from cons with overall rating 1, 2, 3 or 4

The dataset labeled using ruleset one was tested on over 30 different models, resulting in validation accuracy between 22% and 32%. Further inspection of the reviews indicated that the categorization of the nine different sentiments is not appropriate for this dataset. The words and sentences used in reviews don't distinguish enough to justify using so many categories. If we take the examples given in Table 2 (section 3.1), this ruleset assigns the sequences "Some of the clients. Free biscuits and booze." and "Company brand with long history" with different sentiment labels. However, it is hard to argue that the sentiment of these two sequences differs enough to justify assigning them two separate categories. Therefore, ruleset two was chosen for labeling the dataset.

It should be mentioned that the classification of the dataset in two sentiments—where the negative sentiment is assigned to text belonging to the "cons" variable, and positive sentiment to text belonging to the "pros" variable—was considered but discarded as it doesn't depict the dataset well. Looking back at the example from Table 2 (section 3.1), the review with the overall rating 1 states as a pro "Some of the clients. Free biscuits and booze." If binary labeling was used, this review would be assigned a positive sentiment. But if we look at the review as a whole, including for example its headline "If you're into a cut-throat environment, this place is for you", it becomes clear that the reviewer doesn't express a positive attitude toward the company.

The assigned labels are saved in the variable "sentiment", and variables that are not needed for the analysis are removed. Variables left in the dataset are review, job title, firm, and sentiment. Furthermore, all rows with missing values were deleted, leaving 1 337 875 rows in the dataset.

The job title was missing in 9.4% and the headline in 0.01% of rows. Additionally, the variable review had a missing value in two rows.

In the next step, the review text was preprocessed to remove the noise from the data. Double spaces were converted to single spaces, letters were converted to lowercase, and weblinks and special characters were removed. The commonly used text cleaning methods such as lemmatization, stemming, and removal of "stop words" can often have a negative impact on BERT model performance, as they remove contextual information that the model uses to learn. To examine the effect of these methods on the model, two datasets were created: one where these techniques were applied, and one where they were not. The negative impact on the model's performance was noticed, therefore, these techniques were not used.

The dataset was divided into a training and a test dataset. 80% (1 070 518) of observations were assigned to the training dataset, and 20% (267 630) to the test dataset. Since the dataset is unbalanced (Figure 24), it was split in such a way that the distribution of the classes remained the same as in the original.

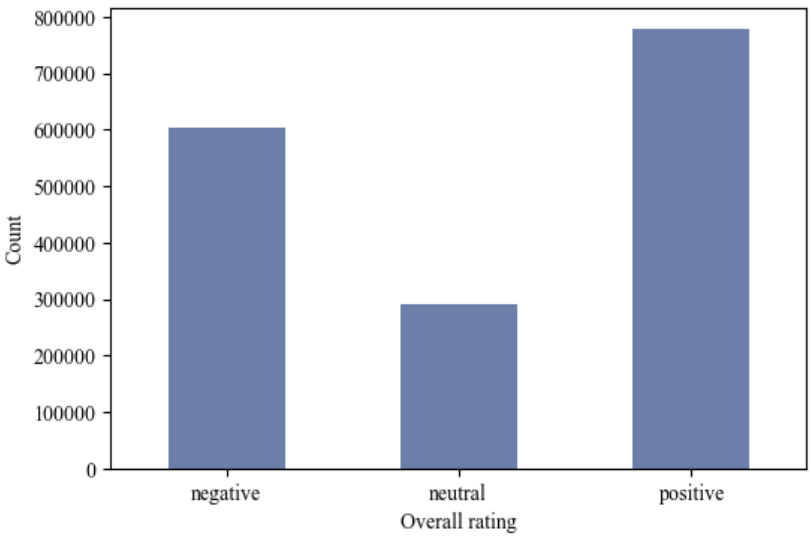

Figure 24 Distribution of the sentiment class

Lastly, the labels were converted to numbers so that 0, 1, and 2 denote a negative, neutral, and positive sentiment respectively, and encoded using one-hot encoding.

4.3 Model

For the sentiment analysis, *BERT base uncased* model from HuggingFace's *Transformers* library was used. This model's maximum input length is 512 sub-word tokens, and it has 12 attention heads, resulting in 110 million parameters. On top of the BERT model, custom layers are added. As previously described, three models for sentiment classification have been trained. The models have the same architecture, but the input to the tokenizer changes.

4.3.1 Tokenization

The BERT model uses the WordPiece tokenizer described in section 2.5.1. This tokenizer returns (Figure 25):

- *input_ids* [length 512] – numerical representation of sub-word tokens in the sequence
- *attention_mask* [length 512] – binary tensor that indicates which tokens should be attended to and which should be ignored. It assigns zero to the padding tokens indicating that they should not be attended to.
- *token_type_id* [length 512] - binary tensor that indicates which sequence a token belongs to. When the input to the model is a single sequence, all values of the tensor are equal to zero. The BERT model uses this token for the Next Sentence Prediction task.

```
input ids:  tf.Tensor(
[  101  7167  1997  2417  6823  1998 25934  2012  2335   102  2470  7155
  2118  1997  5087   102     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0], shape=(90,), dtype=int32)
attention_mask:  tf.Tensor(
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0], shape=(90,), dtype=int32)
token_type_ids:  tf.Tensor(
[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0], shape=(90,), dtype=int32)
```

Figure 25 BERT WordPiece tokenizer output

The input sequence was limited to 90 tokens, and the sequences exceeding this length were truncated. This was done because of the GPU limitations. Figure 26 shows that the number of words in the majority of reviews is less than 90. However, since WordPiece is a sub-word tokenizer, it is hard to estimate how many reviews were truncated.

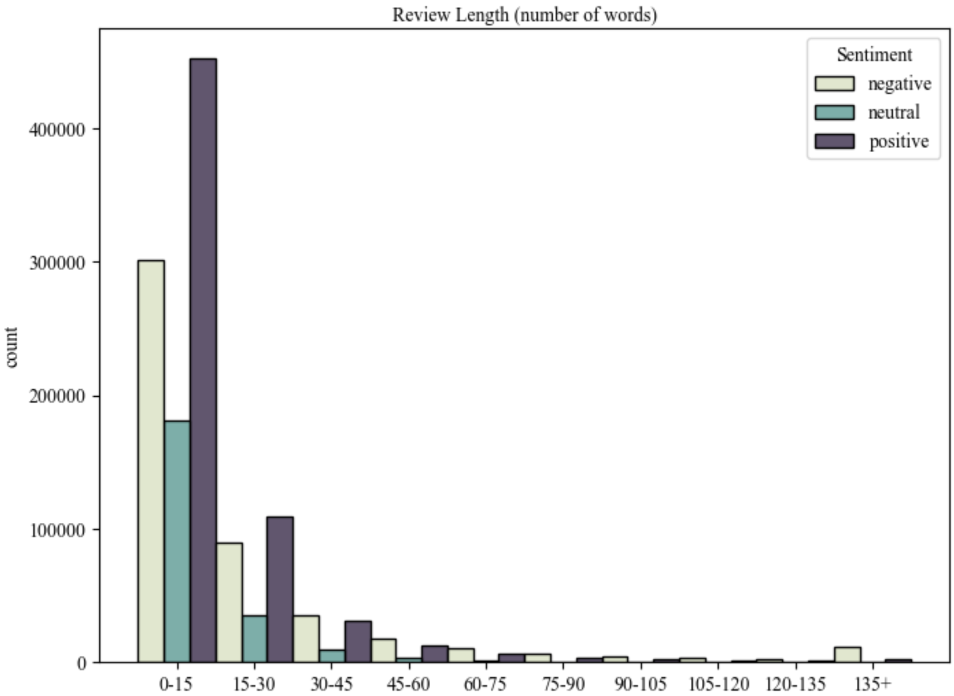

Figure 26 Length of reviews (number of words) across sentiment values. Text length is grouped in ten ranges

Model 1 (single sequence review)

This model takes only the reviews as input. The tokenizer places the [CLS] token at the beginning of the sequence, and the [SEP] token at the end, and adds padding. Since input is a single sequence, all values of the tensor token_type_id are zero, this model doesn't use the Next Sentence Prediction objective.

Input sequence: " i have yet to fine one"

```
[CLS] i have yet to fine one [SEP] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
```

Model 2 (single sequence review + job position + company name)

This model concatenates the review, job position, and company name prior to tokenization. The tokenizer will place the [CLS] token at the beginning, and the [SEP] token at the end of the sequence. Since the input is a single sequence, this model doesn't use the Next Sentence Prediction objective.

Input sequence: "i have yet to fine one senior consultant pwc"

```
[CLS] i have yet to fine one senior consultant pwc [SEP] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
```

Model 3 (sequence pair review–job title + company name)

This model takes the review, job position, and company name as two sequences. The first sequence is the review, and the second sequence is the job title and company name. Given that the sequence pairs are tokenized, the tokenizer will place the [CLS] token at the beginning of the first sequence, one [SEP] token between sequences, and one [SEP] token at the end of the second sequence. Truncation is applied only to the first sequence of sequence pairs that exceed 90 words. This is done to ensure job title and company name are not discarded in the tokenization process. The model adds the Next Sentence Prediction objective since the input is sequence pairs.

Input sequence 1: "i have yet to fine one"

Input sequence 2: "senior consultant pwc"

```
[CLS] i have yet to fine one [SEP] senior consultant pwc [SEP] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
```

4.3.2 Model Architecture

In the process of finding the optimal model architecture and hyperparameters, hundreds of model architectures were tested. However, due to the limited time to conduct the research, as well as hardware constraints, it is possible that the best model was not found. For the purposes of this research, finding the best model is not crucial, therefore this has not affected the outcome and conclusions.

The model architecture and hyperparameter were optimized for Model 1. Then, they were used to train Model 2 and Model 3 as well. Therefore, we can measure how the additional variables and proposed methods for generating the embeddings influence the model's performance. Furthermore, due to the tendency of the BERT model to overfit data easily—even with a large dataset as the one used in this thesis—the focus was to find the architecture that will reduce the overfitting.

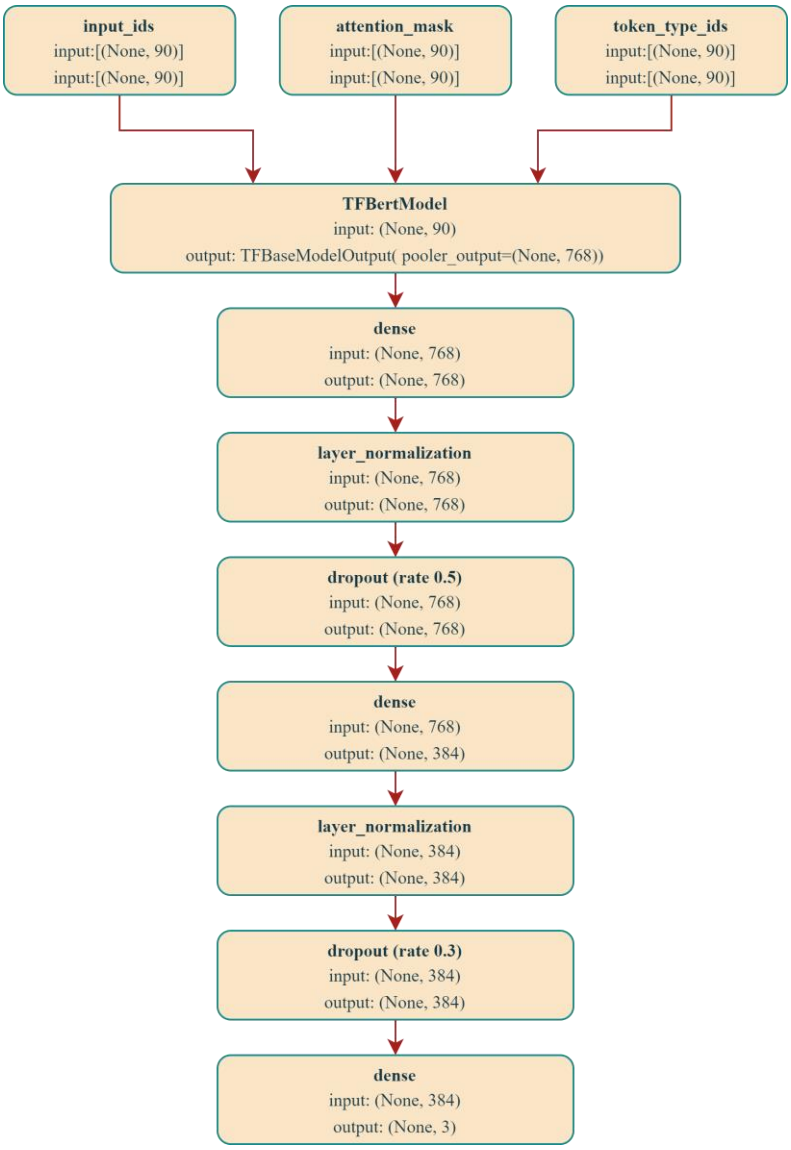

Figure 27 The final model architecture

The model takes input_ids, attention_mask and token_types_ids as input. It has one embedding layer, two fully connected layers, and an output layer. The embedding layer is BERT's pooler_output. This layer transforms the input into sequence embeddings. The output is passed to the first fully connected (dense) hidden layer with 768 hidden units and a $ReLU$ activation function. The output is then fed into the second fully connected hidden layer with 384 hidden unites and a $ReLU$ activation function. The output layer is the linear layer with the $softmax$ activation function, which calculates the probability of three output classes (negative, neutral, and positive). Furthermore, each fully connected layer is followed by layer normalization and dropout (**Error! Reference source not found.**). The model has 110 331 587 parameters (Figure 28). The model architecture with the layers' input and output shapes is available in Appendix A.

4.3.3 Regularization and Optimization

Devlin et al. (2018) provide the range of values for learning rate, and regularization and optimization hyperparameters for fine-tuning BERT which have produced good results on a variety of tasks in their experiments. The given values were used as a starting point for finding the optimal hyperparameters for Model 1. The best results were achieved by:

- L2 regularization with $\lambda = 1e - 3$ is being applied to the weights of both fully connected layers.
- using the dropout rate 0.5 after the first fully connected layer, and 0.3 after the second.
- using batch size 32.
- using the AdamW (Loshchilov & Hutter, 2017) optimization algorithm with learning rate 2e-5.
- using a linear learning rate scheduler with the end learning rate 1e-8.

Furthermore, the objective of the model was to minimize *categorical cross-entropy loss*. Categorical cross-entropy loss measures the difference between the predicted and true probability distribution of the target variable in the multiclass classification problem.

Additional information on hyperparameter tuning, as well as a summary of the observed effects that the adjustments to hyperparameter values and model architecture have on the model performance, can be found in Appendix B. This information is collected to serve as a guideline for further work on the improvement of the model's performance.

5 Evaluation

This chapter provides information on how the models were trained and evaluated. Section 5.2 presents the models' training and evaluation results. This section is followed by the discussion, limitations of the study, and proposals for future work.

5.1 Training and Evaluation

During the training and evaluation, model performance was measured using the following metrics:

- *Precision* is used to measure what portion of the observations that are predicted as positive are actually positive, $precision = \frac{true\ positives}{true\ positives + false\ positives}$
- *Recall* is used to measure what portion of the positive observations are predicted as positives, $recall = \frac{true\ positives}{true\ positives + false\ negatives}$
- *Weighted F1 score* is used to measure model accuracy. It is a weighted average of F1 scores of each class, where weights equals to the proportion of the observations that belong to the corresponding class, and $F1\ score = \frac{2 * (precision * recall)}{precision + recall}$

Devlin et al. (2018) recommend fine-tuning the BERT for 2 to 4 epochs. Fine-tuning the model over more than four epochs leads to overfitting and increases the risk of the model forgetting the knowledge from the lower layers (Howard & Ruder, 2018).

The optimal number of epochs for each of the three models was found using early stopping. This was repeated five times to investigate the training stability. 80% of the dataset was used for training, and 20% for validation. The models behaved consistently over all five trials. The validation loss of all three models started to increase after the third epoch. Weighted F1 score for Model 1 and Model 2 started to decrease after the second epoch, and for Model 3 after the third epoch. This indicates that the optimal number of epochs for Model 1 and Model 2 is two, and three for Model 3, since all three models slowly start to overfit after that. The best parameters for each model were chosen using 5-fold cross-validation.

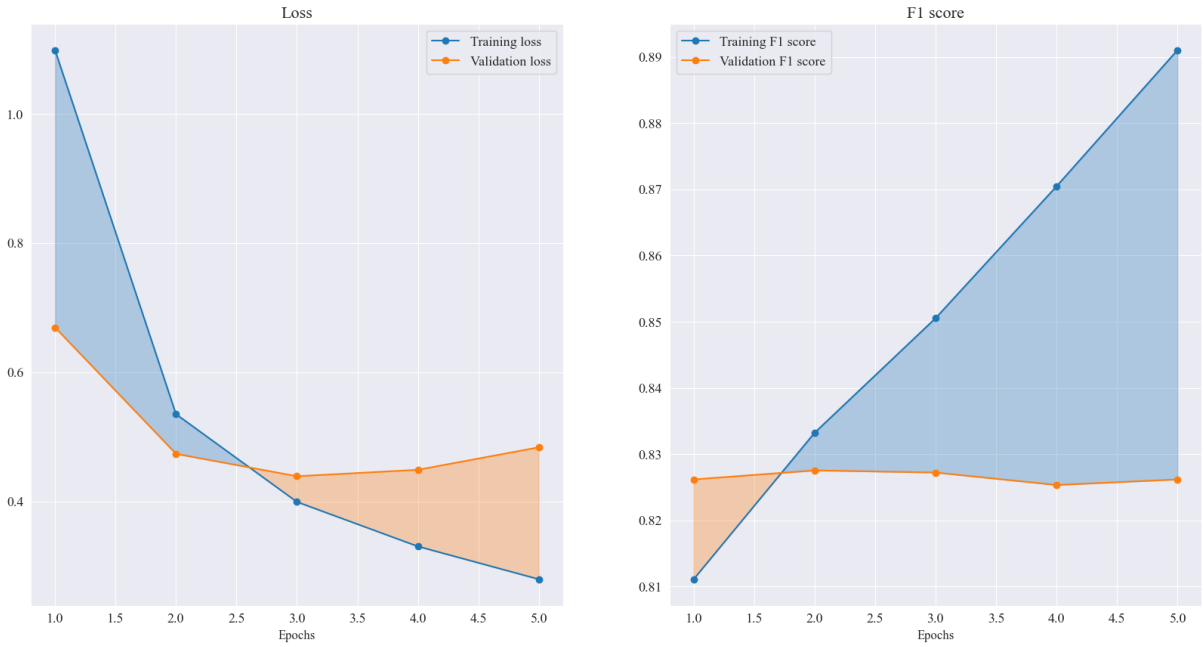

Figure 28 Model 1 - loss (left) and weighted F1 score (right) curve

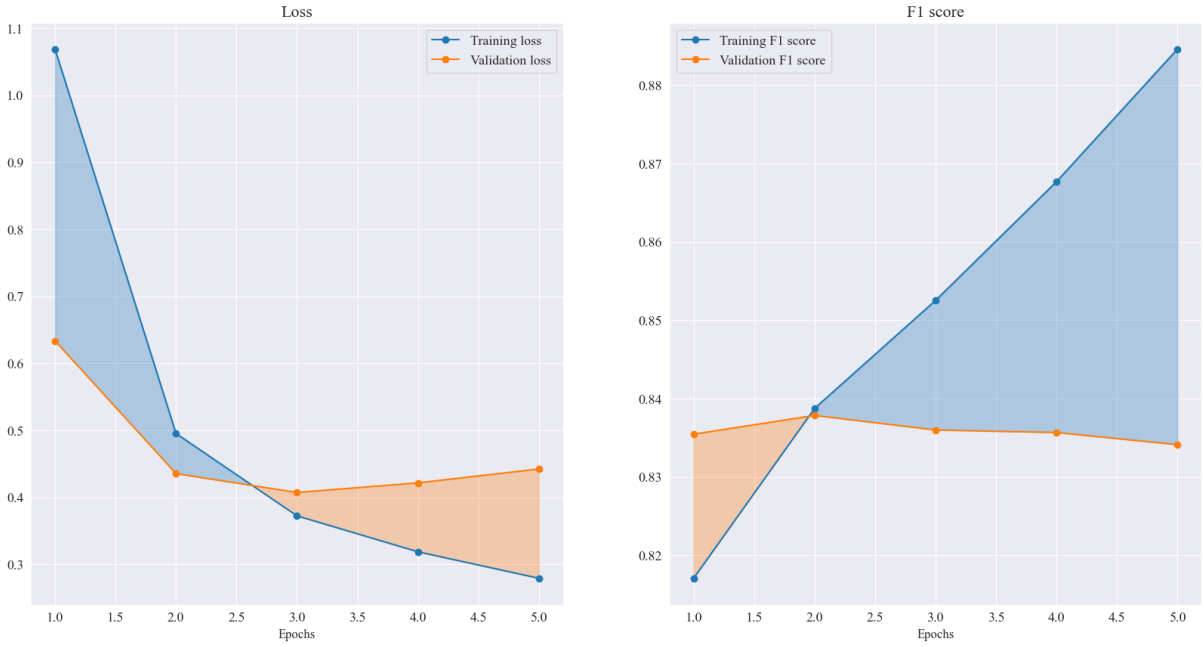

Figure 29 Model 2 - loss (left) and weighted F1 score (right) curve

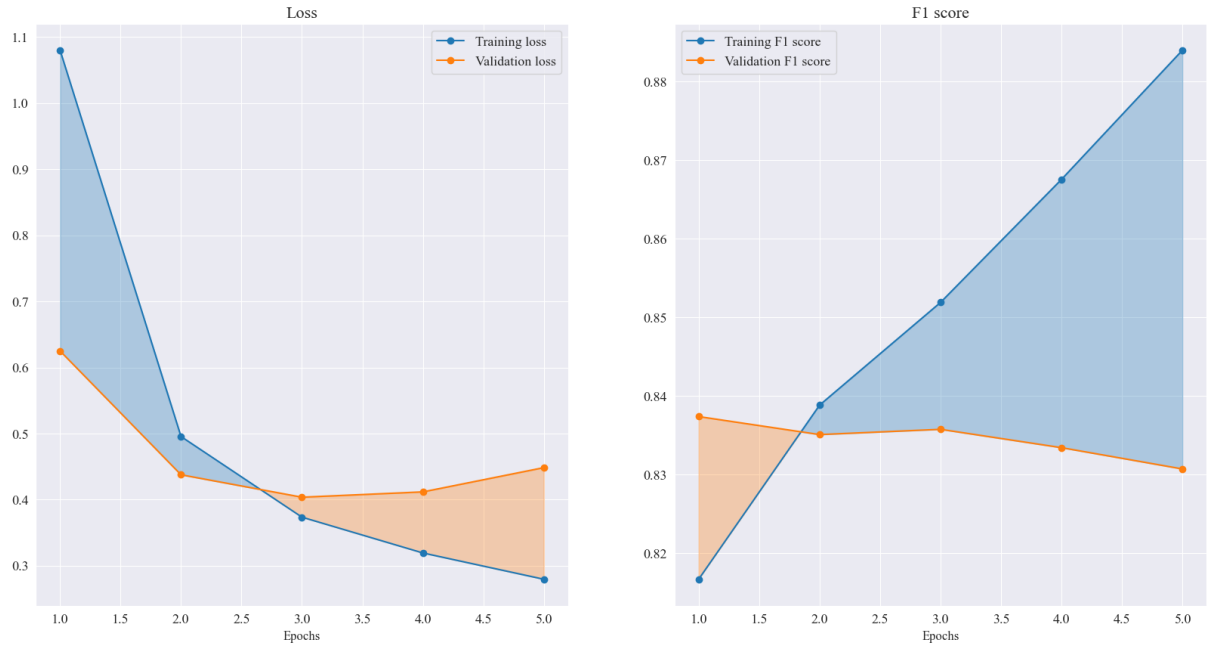

Figure 30 Model 3 - loss (left) and weighted F1 score (right) curve

5.2 Results

Table 4 Evaluation results

Model 1	**weighted F1**	**loss**	**recall**	**precision**
5-fold cross-validation	0.824±0.003	0.448±0.002	0.832±0.001	0.850±0.001
training dataset	0.846	0.402	0.848	0.865
test dataset	0.828	0.447	0.832	0.851
Model 2	**weighted F1**	**loss**	**recall**	**precision**
5-fold cross-validation	0.832±0.004	0.440±0.003	0.838±0.002	0.855±0.002
training dataset	0.852	0.402	0.852	0.869
test dataset	0.836	0.441	0.836	0.856
Model 3	**weighted F1**	**loss**	**recall**	**precision**
5-fold cross-validation	0.838±0.001	0.408±0.000	0.839±0.001	0.850±0.002
training dataset	0.873	0.329	0.872	0.881
test dataset	0.838	0.411	0.837	0.848

Table 5 Model classification report (arrows indicate direction of change compared to Model 1)

Model 1	precision	recall	F1 score	support
Negative sentiment	0.80	0.89	0.84	96754
Neutral sentiment	0.65	0.38	0.48	46468
Positive sentiment	0.92	0.98	0.95	124394
accuracy			0.84	267616
macro avg	0.79	0.75	0.76	267616
weighted avg	0.83	0.84	0.83	267616

Model 2	precision	recall	F1 score	support
Negative sentiment	0.82 ↑	0.87 ↓	0.84	96754
Neutral sentiment	0.64 ↓	0.45 ↑	0.53 ↑	46468
Positive sentiment	0.92	0.98	0.95	124394
accuracy			0.85 ↑	267616
macro avg	0.79	0.76 ↑	0.77 ↑	267616
weighted avg	0.83	0.85 ↑	0.84 ↑	267616

Model 3	precision	recall	F1 score	support
Negative sentiment	0.83 ↑	0.85 ↓	0.84	96754
Neutral sentiment	0.60 ↓	0.51↑	0.55 ↑	46468
Positive sentiment	0.93 ↑	0.96 ↓	0.95	124394
accuracy			0.84	267616
macro avg	0.79	0.77 ↑	0.78 ↑	267616
weighted avg	0.84 ↑	0.84	0.84 ↑	267616

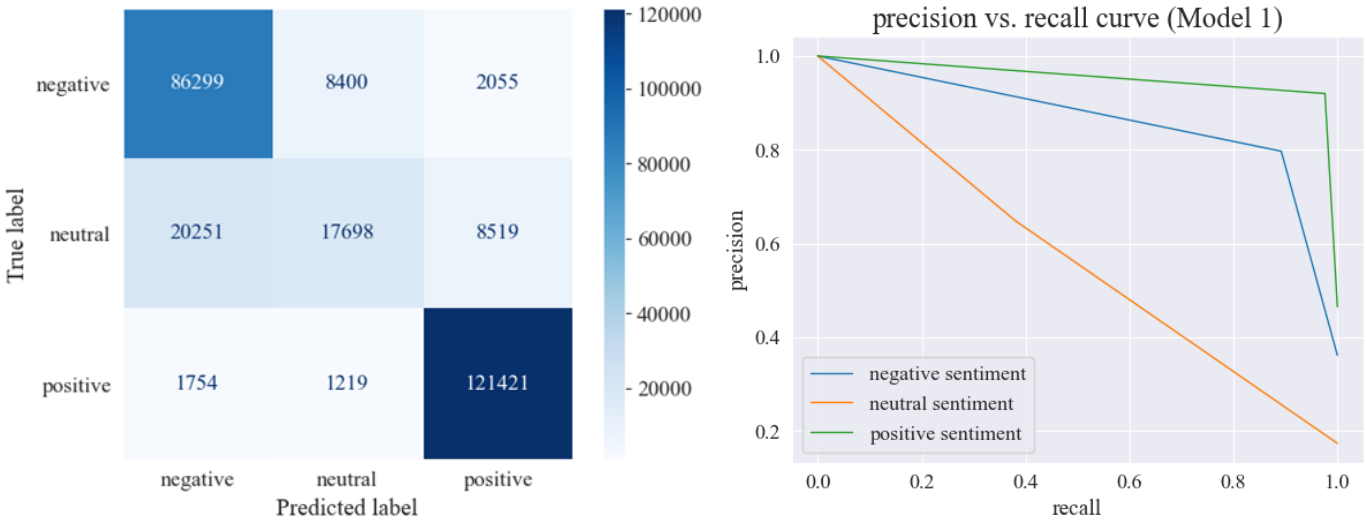

Figure 31 Model 1 - Confusion matrix (left), precision-recall curve (right)

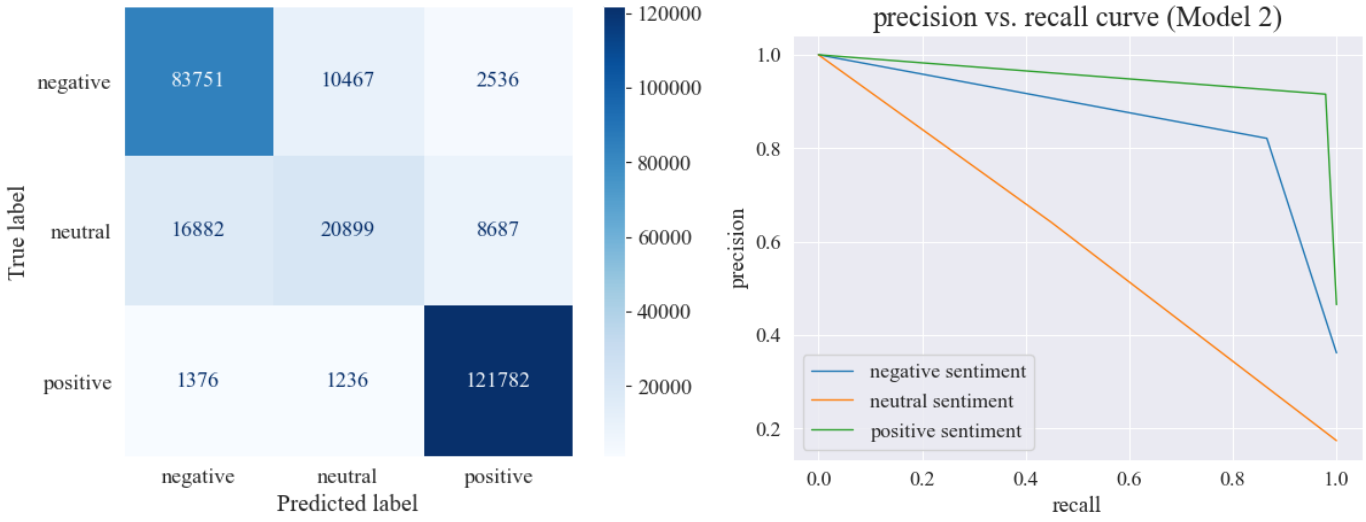

Figure 32 Model 2 - Confusion matrix (left), precision-recall curve (right)

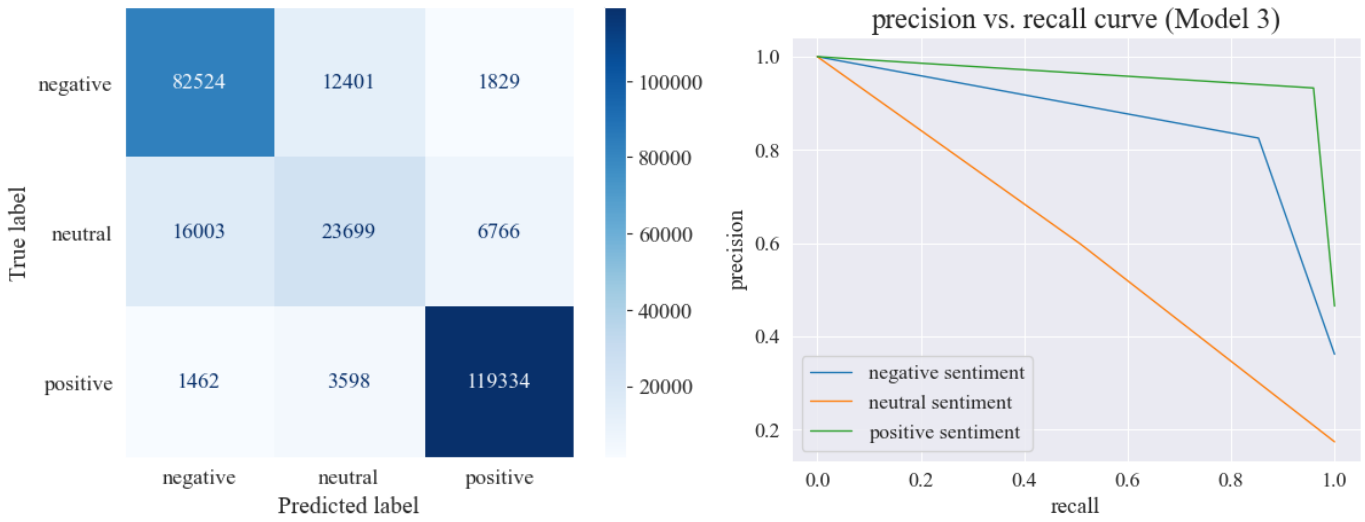

Figure 33 Model 3 - Confusion matrix (left), precision-recall curve (right)

The results show that Model 2 and Model 3 achieved a 0.8–1% higher weighted F1 score compared to Model 1 on the test dataset (Table 4). The performances of Model 2 and Model 3 were very similar, with Model 3 achieving marginally higher accuracy. Furthermore, the results of the 5-fold cross-validation indicate that all models were giving stable predictions.

In the classification report presented in Table 5, we can see that all three models were good at predicting positive and negative sentiments. They all achieved a weighted F1 score of 95% for a positive sentiment and 84% for a negative sentiment. However, all three models were very bad at predicting neutral sentiment. Model 1 performed the worst, with a weighted F1 score of 48%, while Model 2 and Model 3 had a weighted F1 score of 53% and 55%, respectively. The recall value indicates that the models had a very low ability to classify a neutral class correctly. The models found it particularly hard to distinguish a neutral sentiment from a negative one (Figure 31, Figure 32, Figure 33).

The reason for this could be attributed to the labeling rules applied to the dataset described in section 4.2. The neutral class contained statements about both positive and negative aspects of the company. Therefore, it was much harder for the models to learn to recognize it, compared to the other two classes, which contained statements only about one of these aspects. Also, due to the imbalanced distribution of ratings in the dataset, the majority of instances in the neutral class were statements about negative aspects of a company. This could explain why the observations from the neutral class were more often misclassified as negative than as positive.

However, it is interesting that the cause of the improved overall accuracy of Model 2 and Model 3, is their better ability to identify the neutral class. Learning the connections between words used in reviews, job title, and company name helped to reduce the number of cases where a neutral sentiment was falsely classified as negative. This suggests that adding additional features to a review text prior to tokenization resulted in the models learning the embeddings with additional context useful for interpreting the intended sentiment.

The marginal difference in the weighted F1 score between Model 2 and Model 3 suggests that adding the Next Sentence Prediction task to Model 3 didn't result in any significant improvement of the overall accuracy. However, the model achieved a better balance between the precision and recall for each class. Compared to Model 2, it has better capability to correctly identify the neutral class and slightly lower bias toward positive class.

5.3 Discussion

During the study, all the models were trained and tested multiple times, always giving very similar results. This suggests that the difference in the results between Model 1 and the other two models is not a consequence of the random initialization during the models' training. To establish with more certainty what causes this increase in performance, additional research is needed. However, analysis of the results and dataset indicates that the better performance is not caused only by the increased length of the input text.

The median length of the job title is 2 words (mean length 1.94), and of the company name 1 word (mean length 1.56), meaning that in the majority of cases, the input text sequence to Model 2 and Model 3 was longer by only 2–3 words compared to Model 1. Furthermore, Model 2 and Model 3 were more affected by truncation during tokenization. Therefore, Model 1 kept sub-word tokens from the longer reviews—which the other two models discarded—and potentially learned more about the relationship between the words in the review text. However, it was not possible to measure the impact this had on the accuracy of the models.

The better ability of Model 2 and Model 3 to correctly predict the neutral class suggests that the observed improvement could be caused by the choice of the features that enabled the introduction of the informative patterns during the tokenization process. The premise of this research was that there is a relationship between the content of the review, and a company or job position. In section 1.3, the argument was given that if, for example, salaries in a company are low, it is reasonable to assume that most of the reviewers will mention that and give the company a lower rating. Since the same job titles and company names repeat across the dataset, adding them to the review text before tokenization creates similar sub-word tokens that repeat across the input sequences. Model 2 and Model 3 could use this sub-token repetition to learn additional context information, which helps them to identify neutral class more accurately than Model 1.

To see how these models could benefit from this, let's look at the names of the most frequent companies in the dataset. In Figure 21 (section 3.1), we can see that some of the most common companies in the dataset have names which are either acronyms or surnames, such as IBM, SAP, Morgan Stanley, American Express. Since WordTokenizer prioritizes merging the sub-word tokens that appear less often in the vocabulary, pairs such as ['american', 'express'] or ['morgan', 'stanley'] will most likely be merged earlier during the tokenization process (section 2.5.1). This means that the model will produce the review embeddings, which will have a company name (or a significant part of it) incorporated in them. Therefore, it will be able to

learn, for example, for which companies the sequence containing the word “salary” has a negative sentiment, and for which it has a neutral.

Furthermore, the observed effect of adding the Next Sentence Prediction objective on model performance was somewhat unexpected, given the findings of the other research. During the development of BERT, Devlin et al. (2018) found that adding the Next Sentence Prediction objective significantly improved the model performance for a variety of downstream tasks. However, several later studies have shown that it doesn’t always lead to improvement in accuracy, and that in some cases, it can have an opposite effect and hurt the performance of the model (Liu, Ott, Goyal, Du, Joshi, Chen, Levy, Lewis, Zettlemoyer & Stoyanov, 2019; Yang, Dai, Yang, Carbonell, Salakhutdinov & Le, 2019). Therefore, many pre-trained language models developed after BERT don’t use this objective. This study did not find that adding the Next Sentence Prediction task improves the model’s overall accuracy. However, adding this objective has increased the impact of the added features on determining the review sentiment. As a result, the model has become better at classifying observations from the neutral class correctly.

Analysis of the results published in the relevant research leads to the conclusion that—considering the simplicity of implementation, and the low computational complexity it adds to the model—this approach has resulted in decent improvement of the model’s performance. However, the disadvantage of this approach is that the impact of the added features on classification results depends only on the attention score given by the model. Furthermore, concatenating feature vectors to word embeddings gives more flexibility to impact model performance, since it can be done in different stages. Some methods perform it on the input to the model (Liu et al., 2021; Pota, Ventura, Catelli & Esposito, 2020), while others concatenate the output of several models and then pass it to the final output layer (Zhang, Xu, Pang & Han, 2020).

On the other hand, many of the proposed methods for improving embedding accuracy are used only on single context embeddings such as WordToVec and GloVe (Rezaeinia, Rahmani, Ghodsi & Veisi, 2019), and are shown to diminish the performance of the contextual embedding models (Li et. al, 2022). In addition, results published in several studies conducted on social media posts have demonstrated that enhancing word embeddings with semantical features from other elements of the posts, such as emojis (Liu et al., 2021; Pota, Ventura, Catelli & Esposito, 2020) and images (Graesser et al., 2017), can result in much larger improvement of the sentiment classification models. However, these elements are usually not present in the reviews on the employer review platforms.

The findings reported in this thesis indicate that adding features prior to tokenization improves the accuracy of the embeddings, resulting in enhanced performance of sentiment analysis. However, further research is needed to verify whether the hypothesis about the cause of the embeddings’ improvement discussed in this section holds true. The results also show that using sequence pairs can improve the model’s ability to correctly classify the minority class, and improve the balance between precision and recall.

5.4 Limitations

The limited time to conduct the research, along with hardware constraints, has significantly affected the number of trials it was possible to conduct. It has also put a constraint on the length of the model's input. This affected Model 2 and Model 3 more than Model 1, since their input text is longer. However, based on the length of the reviews in the dataset (Figure 26), it can be assumed that the number of input sequences that were truncated was relatively small and could not significantly impact the observed results. To eliminate this uncertainty, the models would need to be trained using full length input sequences.

The same architecture and hyperparameters were used for all three models. This choice was made since there was not enough time to conduct a thorough parameter search for all three models. Therefore, it was not possible to know whether the final models are optimal. In these circumstances, using different parameters could lead to a risk that one model performs better than the other just because better parameters are found, and it would not be possible to compare the results.

5.5 Future Work

Further study could investigate whether the observed improvement in the model was caused by introduction of informative patterns during the tokenization. Furthermore, optimal architectures could be found for the models, and full-length input text used, to better estimate the potential of the used approach. Section 5.4 and Appendix B can be used as a guideline for further work on the optimization of the model architecture. Also, it would be interesting to explore the effects of this approach on other types of pre-trained models, such as ELMo.

During the study, it was observed that labeling the dataset presents a significant challenge for achieving higher model accuracy. This was observed in the other analysis of the employees' review data (Cortinhas, n.d; Jansen, n.d). Further study with more focus on dataset labeling is therefore suggested.

Lastly, the initial proposal for the thesis was to research whether sarcasm detection could improve the sentiment analysis of the employees' reviews. The initial few weeks of work on the thesis were dedicated to the development of a model that would conduct the sarcasm aware sentiment analysis. Unfortunately, this work needed to be interrupted, since it was not possible to finish by the time this thesis needed to be submitted. More details regarding this work are given in Appendix C. Future research could explore whether the described approaches could lead to improved performance of the sentiment analysis.

6 Conclusions

Companies with a good employer reputation have a much higher chance of attracting talented and highly competent workers. Research conducted by Dice (2022) finds that over 78% of the tech professionals find a company's employer brand to be more important than a salary when considering accepting a job offer. Sentiment analysis has become a popular tool for detecting weaknesses in a company's brand by analyzing posts and reviews on online platforms. It faces a lot of challenges, and in the past few years, many new approaches for its improvement have been proposed. However, many of the proposed methods involve building large models that are computationally very expensive to train.

This thesis investigated whether incorporating other elements from the employees' reviews, such as job title and company name, into word embeddings can enhance the performance of sentiment analysis of reviews on the online platform Glassdoor. More specifically, the thesis has tried to answer to following questions:

- Does adding features (e.g., job title and company name) prior to tokenization produce more accurate word embeddings, thereby improving the results of sentiment analysis of employees' reviews?
- Does the Next Sentence Prediction task help the BERT model to learn more accurate word embeddings?

The study included training three different models and comparing their performance. The first model analyzed only reviews, and the second reviews concatenated with job title and company name. The third model analyzed sequence pairs, where the first sequence was the employee's review, and the second a concatenation of job title and company name.

Despite its limitations, this research indicates that the used approach can improve the model performance without significant impact on the computational cost and increase in model complexity. The results show that adding job title and company name prior to tokenization resulted in higher accuracy of sentiment analysis of employees' reviews. However, this study could not provide a definite answer as to whether the observed improvement is a consequence of adding features that enables generation of informative patterns during tokenization. To investigate this hypothesis, further research is needed. In addition, insufficient evidence was found that using sequence pairs improves the models' accuracy. However, adding the Next Sentence Prediction objective improves the models' ability to understand the sentiment of the reviews and correctly predict the minority class. In contrast to earlier research—which found that this objective hurts a model performance—this shows that including this objective could be beneficial in sentiment classification tasks.

References

Alessia, D., Ferri, F., Grifoni, P., & Guzzo, T. (2015). Approaches, Tools and Applications for Sentiment Analysis Implementation. *International Journal of Computer Applications*, vol. 125, no. 3, pp.26-33

Alammar, J. (2018). The Illustrated Transformer, web blog post. Available at: https://jalammar.github.io/illustrated-transformer/ [Accessed 24 April 2023]

Ba, J.L., Kiros, J.R., & Hinton, G.E. (2016). Layer Normalization, preprint, Available online: https://arxiv.org/pdf/1607.06450.pdf [Accessed 24 April 2023]

Bahdanau D., Cho K., & Bengio Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate, preprint, Available at: https://arxiv.org/abs/1409.0473, [Accessed 25 April 2023]

Beechler, S., & Woodward, I.C. (2009). The Global "War for Talent". *Journal of international management*, vol. 15, no. 3, pp.273-285

Bengio, Y., Ducharme, R., & Vincent, P. (2000). A Neural Probabilistic Language Model, in Leen T., Dietterich T., & Tresp V. (eds), *Advances in neural information processing systems*, vol. 13. Available at: https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf

Birjali, M., Kasri, M., & Beni-Hssane, A. (2021). A Comprehensive Survey on Sentiment Analysis: Approaches, Challenges and Trends. *Knowledge-Based Systems*, vol. 226, pp.107134

Cable, D.M., & Turban, D.B. (2003). The Value of Organizational Reputation in the Recruitment Context: A Brand-equity Perspective. *Journal of Applied Social Psychology*, vol. 33, no. 11, pp.2244-2266.

Chapelle O., Schölkopf B., & Zien A. (2010). Semi-Supervised Learning. Cambridge: The MIT Press

Cho, K., van Merri, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP),* pp.1724-1734

Collobert, R., & Weston, J. (2008). A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th international conference on Machine learning*, pp.160-167

Conneau, A., Schwenk, H., Barrault, L., & Lecun, Y. (2017). Very Deep Convolutional Networks for Text Classification. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, vol. 1, pp.1107-1116

Cortinhas S. (n.d.) NLP10 – Transformers. Available at: https://www.kaggle.com/code/samuelcortinhas/nlp10-transformers [Accessed: 12 May 2023]

Dice (2022) Tech Hiring Perspectives. Available at: https://www.dice.com/recruiting/ebooks/dice-tech-sentiment-report/hiring-perspectives.html#Hiring-Perspectives [Accessed 5 March 2023]

Devlin, J., Chang, M.W., Lee, K., & Toutanova, K. (2018). Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding, preprint. Available online: https://arxiv.org/abs/1810.04805 [Accessed 25 April 2023]

DG (2021), Glassdoor Job Reviews, Available online: https://www.kaggle.com/datasets/davidgauthier/glassdoor-job-reviews [Accessed: 9 Mars 2023]

Doshi, K. (2021), Transformers Explained Visually (Part 3): Multi-Head Attention, Deep Dive. *Toward Data Science.* Available online: https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853 [Accessed 24 April 2023]

Ferris, G.R., Perrewé, P.L., Ranft, A.L., Zinko, R., Stoner, J.S., Brouer, R.L., & Laird, M.D. (2007). Human Resources Reputation and Effectiveness. *Human Resource Management Review*, vol. 17, no. 2, pp.117-130

Gers, F.A., & Schmidhuber, J. (2000). Recurrent Nets That Time and Count. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium,* vol. 3, pp. 189-194

Goldie, A., & Hewitt, J. (2022a). Lecture 9: Transformers. CS224n, powerpoint presentation, Stanford University, Winter 2022, Available online: https://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture09-transformers.pdf [Accessed 23 April 2023]

Goldie, A., & Hewitt, J. (2022b). Lecture 10: Natural Language Processing with Deep Learning. CS224n, powerpoint presentation, Stanford University, Winter 2022, Available online: https://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture10-pretraining.pdf [Accessed 25 April 2023]

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT press.

Graesser, L., Gupta, A., Sharma, L., & Bakhturina, E. (2017). Sentiment Classification Using Images and Label Embeddings. Available at: https://arxiv.org/abs/1712.00725 [Accessed 3 Maj 2023]

Habimana, O., Li, Y., Li, R., Gu, X., & Yu, G. (2020). Sentiment Analysis Using Deep Learning Approaches: An Overview. *Science China Information Sciences*, vol. 63, pp.1-36.

He K., Zhang X., Ren S., & Sun J. (2016), Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.770-778

Hewitt, J. (2021). Stanford CS224N NLP with Deep Learning | Winter 2021 | Lecture 9 - Self- Attention and Transformers, [video online], Available at: https://www.youtube.com/watch?v=ptuGllU5SQQ&list=PLoROMvodv4rOSH4v6133s9LFPRHjEmbmJ&index=9 [Accessed: 24 April 2023]

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, vol. 9, no. 8, pp.1735-1780

Howard, J., & Ruder, S. (2018). Universal Language Model Fine-Tuning for Text Classification, preprint, Available at: https://arxiv.org/abs/1801.06146 [Accessed: 12 May 2023]

HuggingFace (n.d). WordPiece Tokenization. Available online: https://huggingface.co/learn/nlp-course/chapter6/6?fw=pt#:~:text=Tokenization%20differs%20in%20WordPiece%20and,vocabulary%2C%20then%20splits%20on%20it. [Accessed 25 April 2023]

Jansen M. (n.d.) NLP | Sentiment Analysis of Company Reviews. Available at: https://www.kaggle.com/code/matthewjansen/nlp-sentiment-analysis-of-company-reviews [Accessed: 12 May 2023]

Joshi, A., Bhattacharyya, P., & Ahire, S. (2017). In: Cambria, E., Das, D., Bandyopadhyay, S., Feraco, A. (eds) A Practical Guide to Sentiment Analysis. Socio-Affective Computing, vol 5. Springer, Cham, pp.85-106

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics,* vol. 1, pp. 655-665

Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks, web blog post. Available online http://karpathy.github.io/2015/05/21/rnn-effectiveness/ . [Accessed: 21 April 2023]

Kannan, S., Karuppusamy, S., Nedunchezhian, A., Venkateshan, P., Wang, P., Bojja, N., & Kejariwal, A. (2016). Big Data Analytics for Social Media, in Buyya R., Calheiros R.N., & Dastjerdi A.V. (eds), *Big Data*, Cambridge: Morgan Kaufmann, pp. 63-94

Kashive, N., Khanna, V.T., & Bharthi, M.N. (2020). Employer Branding Through Crowdsourcing: Understanding the Sentiments of Employees. *Journal of Indian Business Research*., vol. 1, pp. 93-111

Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. (2018). Visualizing the Loss Landscape of Neural Nets. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp.6391–6401

Liu, B. (2017). Many Facets of Sentiment Analysis, in Cambria, E., Das, D., Bandyopadhyay, S., & Feraco, A. (eds) (2017). *A practical guide to sentiment analysis*. Springer

Liu, B., & Zhang, L. (2012). A Survey of Opinion Mining and Sentiment Analysis. In *Mining text data*, Boston: Springer, pp. 415-463

Liu, C., Fang, F., Lin, X., Cai, T., Tan, X., Liu, J., & Lu, X. (2021). Improving Sentiment Analysis Accuracy with Emoji Embedding. *Journal of Safety Science and Resilience*, vol. 2, no. 4, pp.246-252.

Liu L., Liu, J., & Han, J. (2021). Multi-Head or Single-Head? An Empirical Comparison for Transformer Training, preprint, Available online: https://arxiv.org/pdf/2106.09650.pdf [Accessed 24 April 2023]

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized Bert Pretraining Approach, preprint, Available at: https://arxiv.org/abs/1907.11692 [Accessed: 25 April 2023]

Li, Q., Li, X., Du, Y., Fan, Y., & Chen, X. (2022). A New Sentiment-Enhanced Word Embedding Method for Sentiment Analysis. *Applied Sciences*, vol. 12, no. 20, pp.10236.

Loshchilov, I., & Hutter, F. (2017). Decoupled Weight Decay Regularization, preprint, Available online https://arxiv.org/abs/1711.05101 [Accessed 25 April 2023]

Manning, C. (2019). CS224N: NLP with Deep Learning, Winter 2019, Lecture 11 – Convolutional Networks for NLP, Stanford, [video online], Available at: https://www.youtube.com/watch?v=EAJoRA0KX7I&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=12 [Accessed 23 April 2022]

Manning, C. (2021a), Lecture 6: Simple and LSTM Recurrent Neural Networks, CS224n, powerpoint presentation, Stanford University, Winter 2021. Available online: https://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf [Accessed 22 April 2023]

Manning, C. (2021b). Lecture 7: Machine Translation, Sequence-to-Sequence and Attention, CS224n, powerpoint presentation, Stanford University, Winter 2021. Available online: https://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf [Accessed 23 April 2023]

Manning, C. (2022). Lecture 5: Language Models and Recurrent Neural Networks, CS224n, powerpoint presentation, Stanford University, Winter 2022, Available online: https://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture05-rnnlm.pdf [Accessed 22 April 2023]

McKinsley&Company (2022). The Great Attrition is Making Hiring Harder. Are You Searching the Right Talent Pools?, Available online: https://www.mckinsey.com/capabilities/people-and-organizational-performance/our-insights/the-great-attrition-is-making-hiring-harder-are-you-searching-the-right-talent-pools [Accessed 16 April 2023]

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space, preprint, Available online: https://arxiv.org/abs/1301.3781 [Accessed 22 April 2023]

Mohammadi M., Mundra R., Socher R., Wang L., & Kamath A. (2019). Part V – Language Models, RNN, GRU, and LSTM, CS224n, lecture notes, Stanford University, Winter 2019. Available at: https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes05-LM_RNN.pdf [Accessed: 22 April 2023]

Olah, C. (2015). Understanding LSTM Networks, web blog post. Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Accessed 21 April 2023]

Pennington, J., Socher, R., & Manning, C.D. (2014). Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing*. pp.1532-1543

Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep Contextualized Word Representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, pp.2227-2237

Pexman, P. M. (2018). How Do We Understand Sarcasm? Available online: https://kids.frontiersin.org/articles/10.3389/frym.2018.00056 [Accessed 27 April 2023]

Porter, M.E. (1998). Clusters and the New Economics of Competition. *Harvard Business Review*, vol. 76, no. 6, pp. 77-90.

Pota, M., Ventura, M., Catelli, R., & Esposito, M., 2020. An Effective BERT-Based Pipeline for Twitter Sentiment analysis: A Case Study in Italian. *Sensors*, vol. 21, no. 1, p.133.

Pozzi, F.A., Fersini, E., Messina, E., & Liu, B. (2017). Challenges of Sentiment Analysis in Social Networks: An Overview. *Sentiment analysis in social networks*, Boston: Morgan Kaufmann, pp.1-11.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. OpenAI. Available online: https://openai.com/research/language-unsupervised [Accessed: 18 May 2023]

Raschka, S. (2023). Understanding and Coding the Self-Attention Mechanism of Large Language Models From Scratch, web blog post, Available online: https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html [Accessed 23 April 2023]

Rezaeinia, S.M., Rahmani, R., Ghodsi, A., & Veisi, H. (2019). Sentiment Analysis Based on Improved Pre-Trained Word Embeddings. *Expert Systems with Applications*, vol. 117, pp.139-147.

Schaarschmidt, M., Walsh, G., & Ivens, S. (2021). Digital War for Talent: How Profile Reputations on Company Rating Platforms Drive Job Seekers' Application Intentions. *Journal of Vocational Behavior*, vol. 131, pp.103644.

Schuler, R.S., Jackson, S.E., & Tarique, I. (2011). Global Talent Management and Global Talent Challenges: Strategic Opportunities for IHRM. *Journal of world business*, vol. 46, no. 4, pp.506-516.

Schuster, M., & Paliwal, K.K. (1997). Bidirectional Recurrent Neural Networks, IEEE *Transactions on Signal Processing*, vol. 45, no. 11, pp.2673-2681

Singh R. (2017). Utilizing Transformer Representations Efficiently. Available at: https://www.kaggle.com/code/rhtsingh/utilizing-transformer-representations-efficiently [Accessed 5 May 2023]

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Network. *Proceedings of the 27th International Conference on Neural Information Processing Systems,* vol. 2, pp.3104-3112

Taboada, M., Brooke, J., Tofiloski, M., Voll, K., & Stede, M. (2011). Lexicon-Based Methods for Sentiment Analysis. *Computational linguistics*, vol. 37, no. 2, pp.267-307

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A Survey on Deep Transfer Learning, [e-book] STACS 98, pp.270–279, Available Online: https://dx.doi.org/10.1007/978-3-030-01424-7_27 [Accessed 25 April 2023]

Taylor, W.L. (1953). "Cloze procedure": A New Tool for Measuring Readability. *Journalism quarterly*, vol. 30, no. 4, pp.415-433

Tiwari, A. (2019). ELMo Embedding. *Medium*. Available online: https://medium.com/@abhisht85/elmo-embedding-3c7bd0df20d2 [Accessed 25 April 2023]

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R.R., & Le, Q.V. (2019). Xlnet: Generalized Autoregressive Pretraining for Language Understanding, preprint, Available online: https://arxiv.org/abs/1906.08237 [Accessed: 17 April 2023]

van Dongen, T. (2022). Demystifying Efficient Self-Attention. *Toward Data Science*. Available online: https://towardsdatascience.com/demystifying-efficient-self-attention-

b3de61b9b0fb#:~:text=Self%2Dattention%20is%20a%20specific,sequence%20learn%20information%20about%20itself. [Accessed 23 April 2023]

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp.6000–6010

Varsamopoulos, S., Bertels, K., & Almudever, C.G. (2018). Designing Neural Network Based Decoders for Surface Codes, *Quantum Machine Intelligence,* vol. 2, pp.1-12.

Sinoara, R.A., Camacho-Collados, J., Rossi, R.G., Navigli, R., & Rezende, S.O. (2019). Knowledge-Enhanced Document Embeddings for Text Classification. Knowledge-Based Systems, vol. 163, pp.955-971

Wang, X., Jiang, Y., Bach, N., Wang, T., Huang, Z., Huang, F., & Tu, K. (2020). Automated Concatenation of Embeddings for Structured Prediction, preprint, Available online: https://arxiv.org/abs/2010.05006 [Accessed: 5 May 2023]

Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., & Klingner, J. (2016). Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation, preprint, Available online: https://arxiv.org/abs/1609.08144 [Accessed 25 April 2023]

Zhang, A., Lipton, Z., Li, M., & Smola, A. (2021). Dive Into Deep Learning, preprint, arXiv:2106.11342.

Zhang, L., Wang, S., & Liu, B. (2018). Deep Learning for Sentiment Analysis: A survey, preprint, Available online: https://arxiv.org/abs/1801.07883 [Accessed 15 Maj 2023]

Zhang, S., Xu, X., Pang, Y. & Han, J. (2020). Multi-Layer Attention Based CNN for Target-Dependent Sentiment Classification. *Neural processing letters*, vol. 51, pp.2089-2103.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. *In Proceedings of the IEEE international conference on computer vision*, pp. 19-27

Appendix A

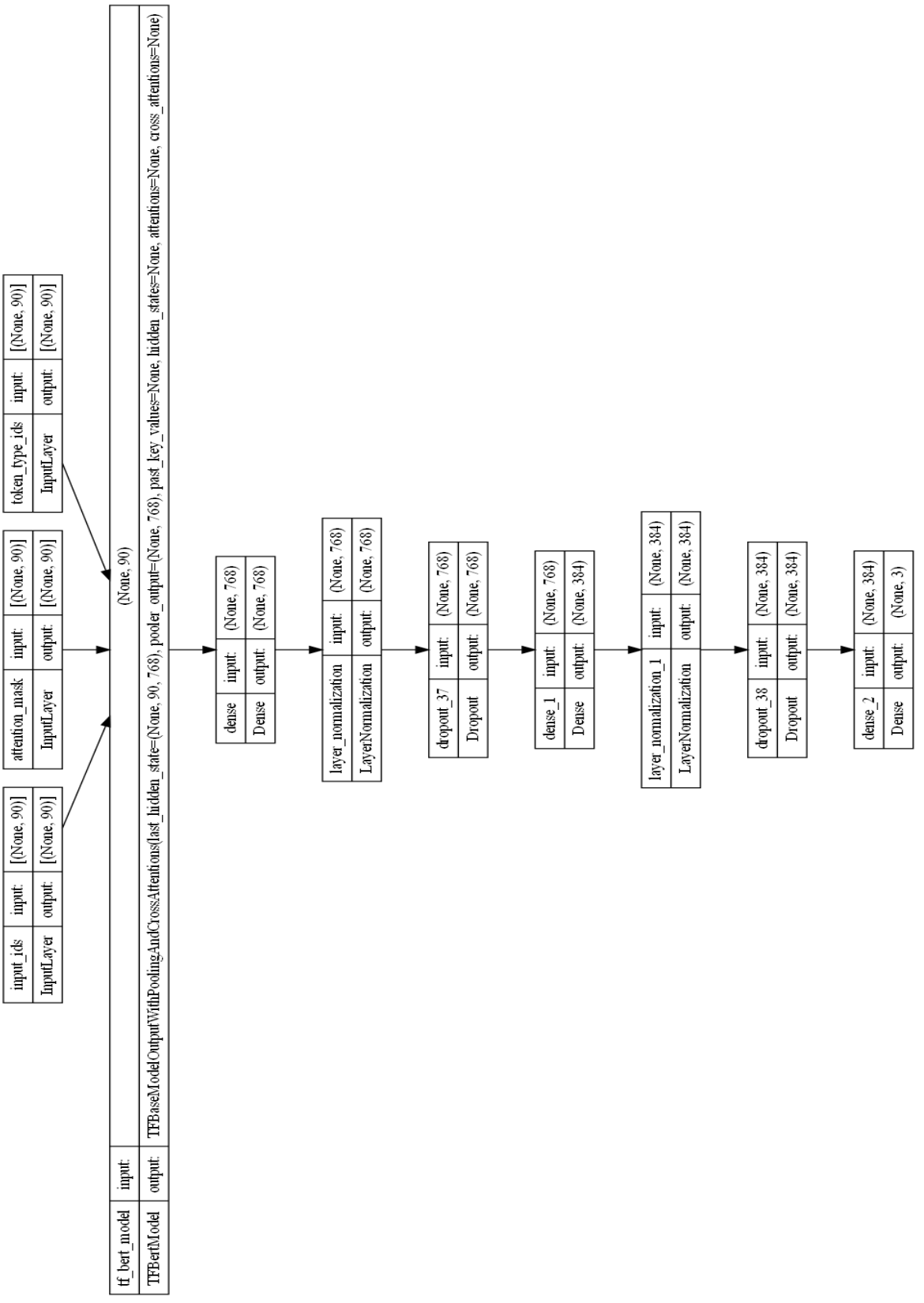

Figure 34 Summary of the final model

Appendix B

Hyperparameter Fine-Tuning – Additional Information

During the model training process, numerous combinations of hyperparameter values and model architectures were tested. Below is a summary of the observed impact that adjustments to the hyperparameter values and model architectures had on the model's performance.

1. Layers and hidden units

The models with 1, 2 and 3 fully connected hidden layers were tested. Models with 1 and 3 hidden layers resulted in a much lower training and validation F1 score, regardless of the number of hidden units. The negative impact was more prominent in the 3-layer network. Furthermore, reducing the number of hidden units of both fully connected layers below the number of hidden units in the final model reduced the training and validation weighted F1 score and increased the loss. Using fewer hidden units in the second layer reduced validation and training weighted F1 score by approximately 0.03. The effect of lowering the number of hidden units in the first layer had a much more severe effect, reducing the training weighted F1 score by approximately 0.07.

In addition, the network with a single LSTM layer was tested, as well as the network with one LSTM layer and one fully connected hidden layer on top of it. Both performed much worse, resulting in a weighted F1 score under 0.74.

2. Regularization

The models with dropout applied only after the first, and the models with dropout applied after both the first and the second fully connected hidden layers were tested. Adding the dropout after the first hidden layer had a noticeable effect on reduction of overfitting. The dropout rate between 0.1 and 0.4 gave relatively similar results, while the larger improvement was noticed with the dropout 0.5. The dropout rate over 0.5 had a negative impact on the model. Furthermore, adding the dropout after the second layer had a small but still positive impact on the model performance. The rate 0.3 seems to be the optimal for this architecture.

Increasing the value of the L2 regularization parameter lambda λ over 1e-3 has reduced the weighted F1 score of both training and validation data significantly, without reducing the distance between them.

3. Learning Rate and Optimization

The learning rates in the range from 1e-6 to 1e-4 were tested. Devlin et al. (2018) state that during the fine-tuning a small learning rate should be used. Depending on the other hyperparameters and architecture, the model had the best performance with leaning rates 2e-5 and 3e-5. Furthermore, adding the learning rate scheduler has significantly improved the model convergence.

Following the recommendation from Devlin et al. (2018) the Adam optimizer with weight decay is used. The AdamW optimizer was chosen because it corrects the errors which exist in the implementation of the weight decay in the standard Adam algorithm (Loshchilov & Hutter, 2017).

4. Other

Both fine-tuning and feature-based approaches were tested on the models that use the pooler_output and the models that use last_hidden_state as the BERT output. Fine-tuning has significantly improved the model's performance. This was expected, considering the dataset has over a million records. Furthermore, fine-tuning the model which uses the last_hidden_state as the output of the BERT resulted the weighted F1 scores 0.76 for the training dataset and 0.74 for the validation dataet. Besides that, this model was more prone to overfitting compared to the model with the pooler_output layer.

Moreover, adding layer normalization after both fully connected layers have significantly stabilized and improved the training.

Lastly, several models were trained using the RoBERTa (Liu et al., 2019) pre-trained model instead of the BERT. In the conducted tests there was no significant difference in the model performance compared to BERT.

Appendix C

Sarcasm detection

Detection of sarcasm in text is a complex task even for humans. While in verbal communication, we can relatively easily recognize someone is being sarcastic through body language, intonation, or prior knowledge regarding a person's attitude about a topic (Pexman, 2018), in written communication most of these hints are not present. Due to its complexity, recognition of sarcasm is considered to be one of the hardest tasks in the NLP field. A challenge of sentiment analysis is that words commonly used to express a positive sentiment, are used sarcastically to express a negative sentiment.

The field has gained a lot of attention in the past few years, and many models have been developed. However, due to the lack of large, labeled datasets, the majority of research is narrowed to the detection of sarcasm in Twitter posts, Reddit posts and News Headlines.

The original idea for the thesis was to develop a model that would incorporate the knowledge of sarcasm—learned on the combination of Twitter posts, Reddit posts and News Headlines datasets—to the sentiment-aware embeddings—learned on the employees' reviews. This would produce the sarcasm-aware model for sentiment analysis of the employees' reviews. The two possible approaches were considered:

1. training two separate models—one for sarcasm classification, and another for sentiment classification—and utilizing the attention mechanism to incorporate the knowledge about the sarcasm from the sarcasm-aware embeddings into the sentiment-aware embeddings.
2. fine-tuning the BERT model for the task of sarcasm detection as the intermediate task to create sarcasm-aware sequence embeddings and fine-tuning the same BERT model again (or training a Bi-LSTM model) for sentiment analysis as the final task.

During the work on the thesis, the sentiment classification model and the sarcasm classification model have been developed. Unfortunately, it was realized that the amount of time needed for coding, training, and testing the final model was longer than anticipated, and that the work cannot be finished by the thesis submission deadline.