FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Sentiment Analysis using Domain Specific Adapters
Student:	Bc. Lukáš Langr
Supervisor:	Ing. Daniel Vašata, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of winter semester 2022/2023

Instructions

Sentiment analysis is an approach that aims to extract the polarity of a given text. Such polarity may, for example, correspond to a positive or negative review of some product. The problem is that sentiment analysis is a very domain-specific task and fine-tuning a whole model for a domain takes a lot of computational power. The aim of this thesis is to use light-weight domain-specific adapters on top of a frozen general base model to achieve similar performance as a fine-tuning approach would while using a minimum amount of trainable parameters.

1) Review the state of the art approaches for sentiment analysis using transfer learning. Focus on the usage of domain-specific adapters.

2) Experiment with different architectures of the base model and the adapters. Compare the achieved results with a fine-tuned model baseline.

3) Propose a direction for further improvement of selected approaches.

Electronically approved by Ing. Karel Klouda, Ph.D. on 12 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Sentiment Analysis using Domain Specific Adapters

Bc. Lukáš Langr

Department of Applied Mathematics
Supervisor: Ing. Daniel Vašata, Ph.D.

February 10, 2022

Acknowledgements

I would like to immensely thank my girlfriend Julie for believing in me and for supporting me during the creation of this thesis.

Declaration

In Prague on February 10, 2022 . . . .

Citation of this thesis

Langr, Lukáš. *Sentiment Analysis using Domain Specific Adapters.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Ve zpracování přirozeného jazyka v poslední době dominují velké předtrénované modely vyžadující mnoho výpočetního výkonu na přizpůsobení se konkrétní úloze. V této práci je navržena jiná metoda přenášení znalostí zvaná doménově specifické adaptéry pro úlohu analýzy sentimentu. Adaptované modely jsou porovnány s fine-tune-ovanou baselinou v několika experimentálních scénářích a jejich výkonnost je srovnatelná s mnohem většími modely, ikdyž jsou mnohem méně výpočetně náročné. Tento přístup se jeví být použitelnou alternativou k velkým modelům v prostředích s nízkým výpočetním výkonem.

Klíčová slova Analýza sentimentu, Transfer learning, Doménově specifické adaptéry, Recenze filmů, Recenze aerolinek

Abstract

Natural language processing has become a domain of large pre-trained models requiring a great deal of computing power to adjust to a custom task. In this work a different transfer learning method of domain specific adapters is proposed for the task of sentiment analysis. The adapted models are compared to a fine-tuning baseline in multiple experimental scenarios and their performance is comparable to considerably larger models while being much less computationally intensive. This approach looks to be a viable alternative to large models in lower computing power environments.

Keywords Sentiment analysis, Transfer learning, Domain specific adapters, Movie reviews, Airline reviews

Contents

List of Figures

List of Tables

Introduction

The growth of user-generated reviews and opinions has been going nowhere but up in the last decade or two. That has created a huge demand for internet based businesses to develop efficient and accurate tools to computationally evaluate this content and extract valuable knowledge about their customers.

Therefore, sentiment analysis became a prominent direction of the natural language processing field. The methods employed by sentiment analysis have been steadily leaning away from hand made lexicons with heuristic rules towards machine learning approaches. With the computing power rising and hardware prices falling, neural networks became the go-to machine learning technique used by sentiment analysis researchers.

Nowadays, laptops and smartphones contain sufficient performance to run inference of neural networks in real time and to train some smaller personalized networks as well. However, these devices still do not possess adequate power to train a whole language model from scratch. As a result, this creates a need for using an already trained model.

To expand on that, computing power is not the only issue. Companies with international customer bases and numerous domains of business are trying to understand and evaluate multiple different types of datasets. These datasets might not be large or diverse enough for training their own dedicated model. This creates a need for general language models whose knowledge can be adapted to the use on low resource datasets and domains.

Transfer learning has been trending in the recent years. In the field of computer vision, transfer learning has enabled the pre-training of massive convolutional networks on enormous amounts of images e.g. VGG-19 [12] or ResNet-50 [13]. Subsequently, these models can be shared to be fine-tuned to particular tasks. This approach is now becoming prevalent in the field of natural language processing, notably with the arrival of the Transformer in 2017 [7] and its derivatives BERT [8], GPT-2 [14] and GPT-3 [15].

Nevertheless the aforementioned models are still too computationally demanding to possess the ability to be fine-tuned on a consumer's device. Consequently, light-weight adapters present a potential solution. Adapters proposed by Bapna et al. in [10] can be used to solve both previously mentioned problems. Firstly, they are light-weight and for that reason training them to customize an already pre-trained model is computationally inexpensive. Secondly, while keeping the large base model the same, having multiple adapters for high and low resource domains makes it very straightforward to swap them if necessary.

Chapter 1

Goals

The aim of the theoretical part of this thesis is to research the state-of-the-art methods for the use of transfer learning in sentiment analysis or natural language processing in general. Adapting smaller parts of machine learning models to different in-and-out of domain datasets represents the main focus of the research. This approach is then put into contrast with fine-tuning of entire models.

The implementation part aims to conduct experiments with a proposed model using light-weight adapters, in order to utilize transfer learning on multiple datasets. These experiments should provide evidence that the performance of the proposed model is comparable to the fine-tuning baseline.

Both of these parts aim to either prove or disprove whether a model using light-weight adapters can be considered a worthy alternative to fine-tuning in certain scenarios.

Chapter 2

Sentiment analysis

2.1 What is Sentiment Analysis

Sentiment analysis, also referred to as Opinion mining, is natural language processing (NLP) task of classifying opinions of authors of given texts. [16] These opinions can be a polarity - whether or not the author thinks **positively** or **negatively** about the subject or it might be a scale *1-5 stars*, *x out of 10* etc.

Sentiment analysis is divided into many subcategories and approaches. Generally, there are three levels of analysis and two approaches which will be explained in the following chapters. [17]

2.2 Levels of Analysis

Sentiment analysis is mostly performed on three levels: document level, sentence level and aspect level. [17]

Document Level

The document level sentiment analysis attempts to classify whether a whole document voices a positive or negative opinion about a single subject. Usually, this is the case of some product reviews or company feedback. The presumption that the analyzed document expresses one coherent sentiment about an individual entity is very important. As a result this technique is inadequate for texts evaluating multiple entities. [17] [16]

Sentence Level

A role of the sentence level is to determine the opinion of each sentence in a document. A sentence can be either positive or negative, or it have a neutral sentiment, meaning it is simply stating a fact without expressing any emotion about it. These sentence classes can also be labeled *objective* or *subjective* [18]. Although, subjectivity and opinion must not be confused for the same thing. An objective sentence can carry sentiment e. g. “The new computer we bought is broken.” compared to a subjective sentence without any sentiment “I think this building is old.” [17] [16]

Aspect Level

The most complex sentiment analysis task is aspect level classification. It aims to discover the author’s opinion about each entity in the text. For example in a sentence “I love apples.” there is only one aspect, “apples”, about which an opinion is expressed. In this case “apples” is the *opinion target.* Using opinion targets gives us a much finer understanding about all sentiments being conveyed in a sentence. [17]

A single sentence can express multiple sentiments about multiple targets, for example “Although the food is horrible, I still like the atmosphere in this cafe.” If this review was analyzed using the sentence level approach, it would be really hard to say if it is completely positive or completely negative. It states that the “food” is *negative* but the “atmosphere” is positive. As a result for sentences like the above it is clear that this approach is necessary to extract maximum knowledge. [17] [16]

Chapter 3

Language Classification Models

3.1 Lexicon-based Approach

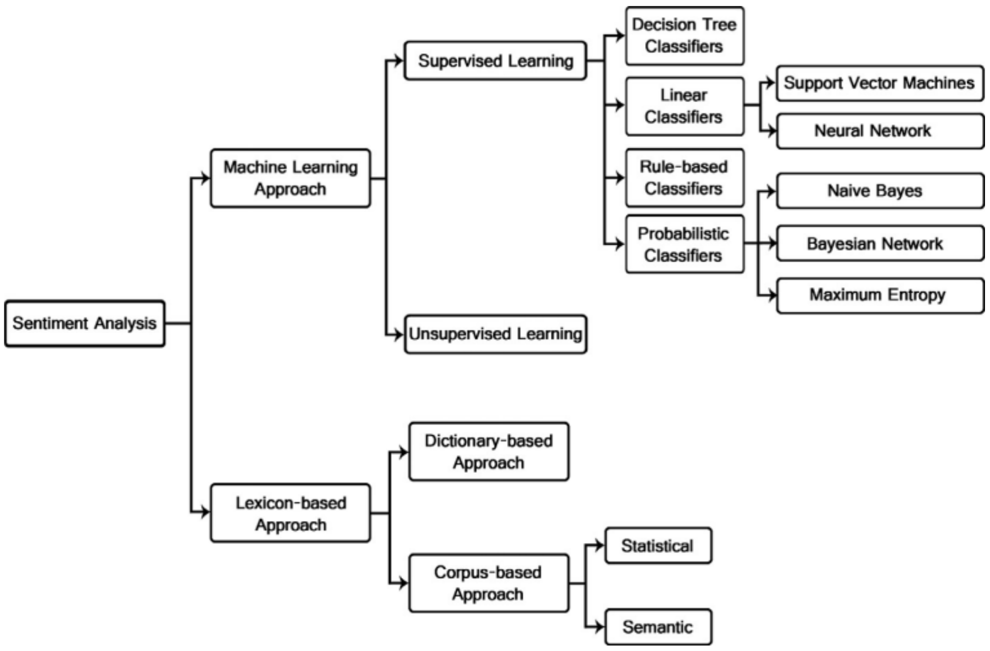

Figure 3.1: Tree of sentiment analysis techniques [1].

The most straight-forward approach for classifying texts is to use some kind of indicator words. For example “good”, “great” and “wonderful” are positive words and “bad”, “horrible” and “poor” are negative ones. Consequently words expressing positive and negative sentiments are compiled into *sentiment lexicons*. Sentiment words are important for classification but by no means sufficient. [17]

The following section outlines some issues a lexicon-based approach can run into:

1. A word can mean the opposite sentiment in another domain or a different sentence context. For example, the word *suck* is used in a negative connotation as it is in a sentence “This camera sucks”, but in another domain “This vacuum cleaner really sucks” it means the exact opposite. [17]
2. The existence of sentiment words in a sentence does not necessarily mean there is an opinion being expressed. For example, in questions and conditional sentences, “Can you tell me which Sony camera is good?” or “If I can find a good camera in the shop, I will buy it.” Even though both of these texts contain the word “good” there is no evidence of it meaning that the sentiment of the sentence is positive. [17]
3. Sarcastic sentences are notoriously very hard for lexicons to deal with because they contain the opinion words of one polarity group but their meaning is exactly the opposite, for example “What a great car! It stopped working in two days.” [17]
4. Some sentences may imply the negative opinion without the use of sentiment words. For example “This washer uses a lot of water.” conveys a negative opinion about the product since a high water consumption is a bad thing but it will not be categorized properly just by searching for negative opinion words. [17]

Due to these shortcomings, the focus has been shifting from lexicon-based sentiment analysis towards machine learning approaches in the recent years. [17] [16]

Examples

AFINN lexicon

AFINN is a lexicon developed by Finn Årup Nielsen. It consists of around 2500 words and phrases which produces a sentiment analysis score for a given sequence. [19]

VADER

VADER (Valence Aware Dictionary and sEntiment Reasoner) is an open-source lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media. [20]

3.2 Machine Learning Approach

Machine learning (ML) has seen a huge boom in the last decade with the improvements in computing power. In consequence this is why it became a viable strategy for sentiment classification. [21]

In general, machine learning focuses on creating mathematical models and feeding it data for it to learn to recognize patterns. [21] There are two important approaches to machine learning:

Supervised learning = a model learns to a function that maps an input to an output based on input-output pairs [22].

Unsupervised learning = a model learns patterns in the input even though no explicit feedback is supplied [22].

ML sees sentiment analysis as either binary (positive or negative) or n-ary (varying degrees of positivity, negativity and neutrality) supervised classification problem.

Classification Workflow

Every classification task follows these steps:

1. Load input data.
2. Split input data into training and testing subsets.
3. Select models and their parameters.
4. Train models using only the training dataset.
5. Evaluate trained models using the testing dataset.

Ever since, Pang and Lee introduced a new approach for solving Sentiment analysis tasks in 2002 [23], the traditional approach of manually creating lexicons gained an alternative of using supervised machine learning algorithms on texts. This revolutionary idea increased the performance of classifiers and opened a door for a new branch of research. [23]

There are multiple challenges to be tackled when classifying texts using ML. The first is how to represent text as numbers. Since ML models expect vectors of numbers as their input, they are incompatible with text. Techniques how to represent text will be discussed in detail in the following section.

The next challenge is picking a model. There are several good options in both traditional statistical models as well as modern deep learning neural networks (NNs). Most notable of those is the **Transformer** which is the model of choice of the practical part of this thesis.

Representing Texts

Text is an example of unstructured data. Machine learning models require vectors of numbers as inputs so it is necessary to transform words into numbers. [24] Since languages are messy and multitudinous words are redundant, some preprocessing is always useful. Preprocessing can have many forms, here are the most common ones:

- Tokenization – converting sentences to lists of words,
- Removing punctuation – converting sentences to lists of words,
- Removing stop words – words such as ”the”, ”a” or conjunctions and prepositions which do not convey any meaning,
- Stemming – reducing words to a root,
- Lemmatization – reducing different forms of the same word to a lemma, e.g. reducing ”was” and ”were” to ”be”. [25]

When the texts are preprocessed, the next step is to extract features.

Bag of Words

The simplest way to represent a text as a vector of numbers is a bag of words (BOW). A sentence s is transformed to a vector v of size n, where $v[i]$ is the count of the i-th word of the corpus in s and n is the size of the corpus. [24] [26]

For example the sentence: ”John likes to watch movies and Mary likes them too.” in a corpus of [a, and, John, likes, Mary, movies, something, the, them, to, too, watch] would be represented as [0, 1, 2, 1, 1, 0, 0, 1, 1, 1]. [24] [26]

TF-IDF

TF-IDF stands for Term Frequency – Inverse Document Frequency. It is a greatly used technique for transforming a set of documents, also called corpus, to a set of vectors of numbers representing said documents. The TF-IDF values are products of two quantities. [24] [27]

TF

The first is term frequency (tf). It measures how much is a word used in a document. There are many ways how tf can be produced. [27] The most common formulas are:

$$\mathrm{tf}(w, d) = \begin{cases} 1 & \text{if } w \in d, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathrm{tf}(w,d) = \mathrm{f}_{w,d},$$

where $\mathrm{f}_{w,d}$ is the number of occurrences of w in d,

$$\mathrm{tf}(w,d) = \log\left(1 + \mathrm{f}_{w,d}\right),$$

$$\mathrm{tf}(w,d) = \frac{\mathrm{f}_{w,d}}{\sum_{w' \in \mathcal{D}} \mathrm{f}_{w',d}}.$$

IDF

The second is inverse document frequency (idf) which quantifies how common or rare a word is in the whole corpus. [27] Its values can be calculated like this:

$$\mathrm{idf}(w,\mathcal{D}) = \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : w \in d\}|}$$

$$\mathrm{idf}(w,\mathcal{D}) = \frac{|\mathcal{D}|}{1 + |\{d \in \mathcal{D} : w \in d\}|}.$$

The final TF-IDF value for a word w and document $d \in \mathcal{D}$ is a product of term frequency and inverse document frequency

$$\text{tf-idf}(w,d) = \mathrm{tf}(w,d) \cdot \mathrm{idf}(w,\mathcal{D}). \tag{3.1}$$

Word2vec

In 2013, Mikolov at el. introduced a new way of representing words in computers [2]. They created a two-layer neural network (NN) which takes text as input and produces n-dimensional vectors called word embeddings. What they discovered is that the neural net preserves syntactic and semantic word similarities without requiring labeled data as input (it is unsupervised). E.g. if high dimensional vectors are trained on a large amount of data, the factual relation between two words like Berlin is a capital city of Germany can be applied similarly to France just by using vector arithmetic

$$\mathrm{vec}(\text{“Berlin”}) - \mathrm{vec}(\text{“Germany”}) + \mathrm{vec}(\text{“France”}) = \mathrm{vec}(\text{“Paris”}).$$

Word2vec can work in two different modes. CBOW – continuous bag of words is method when the NN is trying to predict the target word from context and Skip-gram when it is trying to predict the context from the target word. Both architectures can be seen in 3.2. [2]

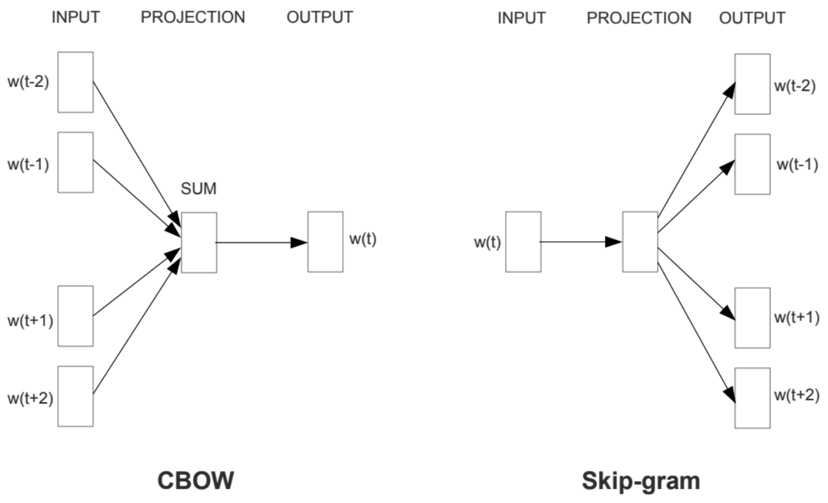

Figure 3.2: Two model architectures of Word2vec [2].

Statistical Models

Naive Bayes

Naive Bayes is a probabilistic model using the Bayes theorem. It was used by Pang and Lee in the first ML sentiment analysis paper [23].

$$\mathrm{P}(C_k|x) = \frac{\mathrm{P}(C_k)\,\mathrm{P}(x|C_k)}{\mathrm{P}(x)} \tag{3.2}$$

It aims to calculate the conditional probability that an instance represented by a vector of features x is from a class C_k based on the features x and it used the assumption that all the features x_i are mutually independent. This is not usually the case in the real world hence the name **Naive** Bayes. [22]

Deep Learning

With the improvement of computing power, the deep learning neural networks have become popular in almost all machine learning branches. A simple neural network consists of layers. There is an input layer, output layer and a possibility of many hidden layers, as can be seen in 3.3.

In a **fully connected** neural network, each neuron is connected to the outputs of all the neurons from the previous layer. Those connections are multiplied by weights and passed into an activation function.

There are many activation functions to choose from. For example the linear function, ReLU and the sigmoid function are all very useful.

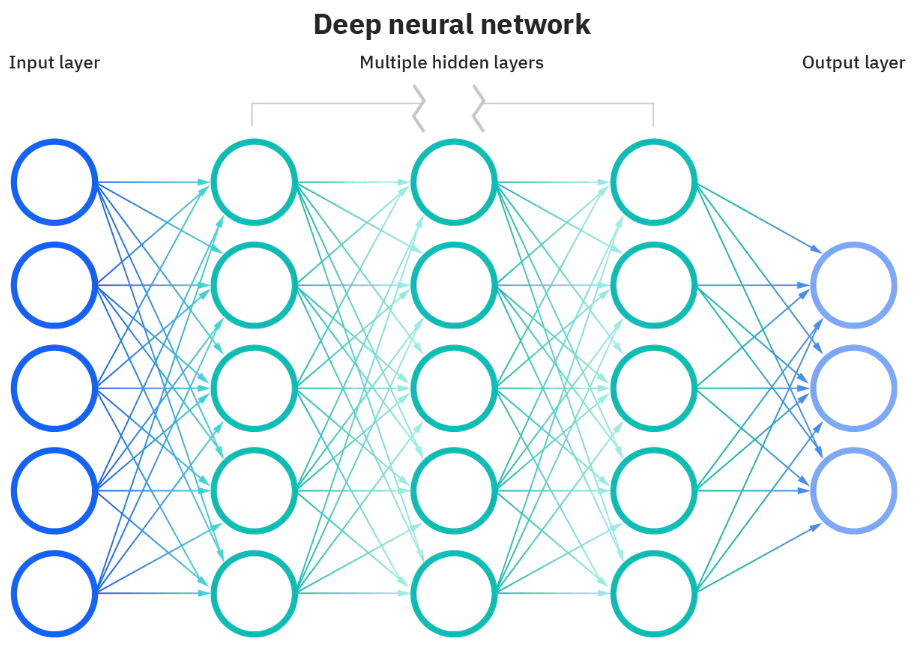

Figure 3.3: A deep feed-forward neural network [3].

Linear function is defined as

$$\text{linear}(x) = x \tag{3.3}$$

meaning the input is the output. ReLU stands for Rectified Linear Unit and it can be seen in figure 3.4. It is defined as

$$\text{ReLU}(x) = \max(0, x). \tag{3.4}$$

ReLU is non-linear for $x < 0$ and linear $x > 0$.

Another useful activation function is the sigmoid function also known as the logistic curve. Its plot can be seen in figure 3.5. The sigmoid function is defined as follows

$$\sigma(x) = \frac{1}{1 + e^{-1}}. \tag{3.5}$$

The simplest neural network is the fully connected or feed-forward network (FFN) which can be seen in 3.3. As the name suggests, FFNs have only forward connections between nodes.

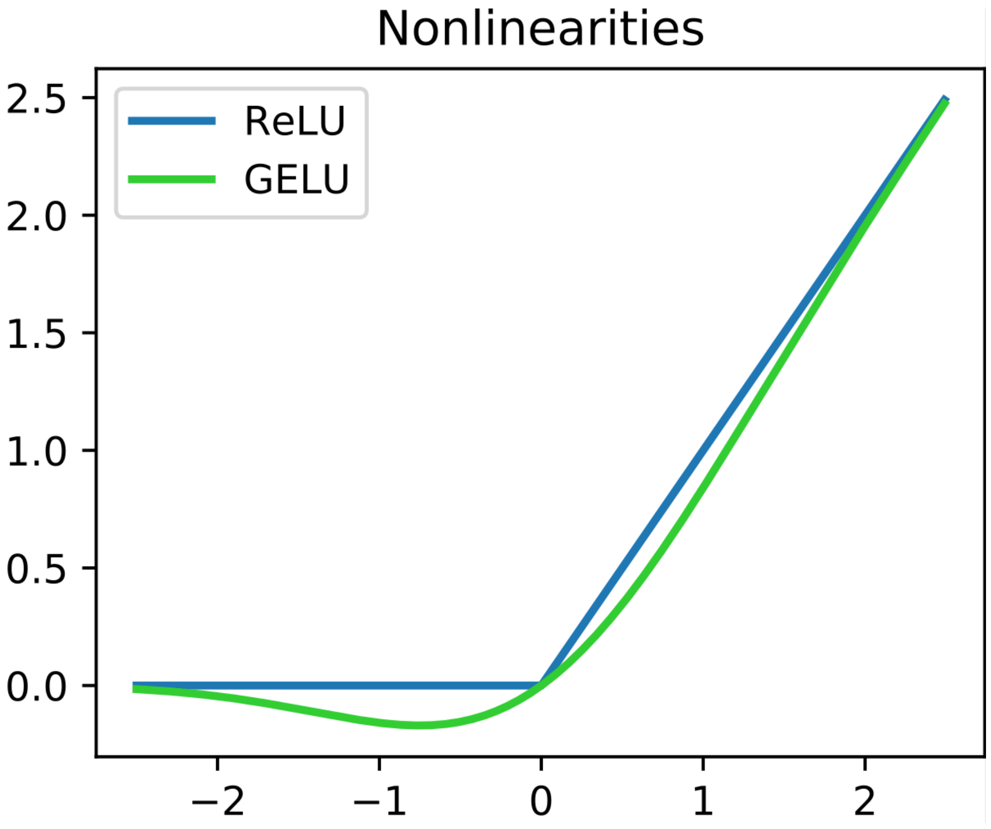

Figure 3.4: Plot of ReLU and GELU

RNNs and CNNs

Two types became particularly useful in the natural language processing field - recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

In comparison to feed-forward networks, recurrent neural networks allow for backward connections, giving them the ability to keep state which can be seen as an example in image 3.6. This enables them to be **sequence to sequence models** meaning that inputs and outputs are time series.

Recurrent networks are strictly sequential which makes them very slow to train.

On the other hand convolutional neural networks are much more lightweight in terms of computational complexity. They are primarily used for images which they can learn to extract simple features from instead of using the whole 2D array as inputs which would be inefficient. This can be applied to text data in a similar fashion.

The words are represented as n dimensional vectors and a weight matrix is slid horizontally across the sentence with a stride of 1. This filter matrix which usually takes a maximum or average value from a window extracts features

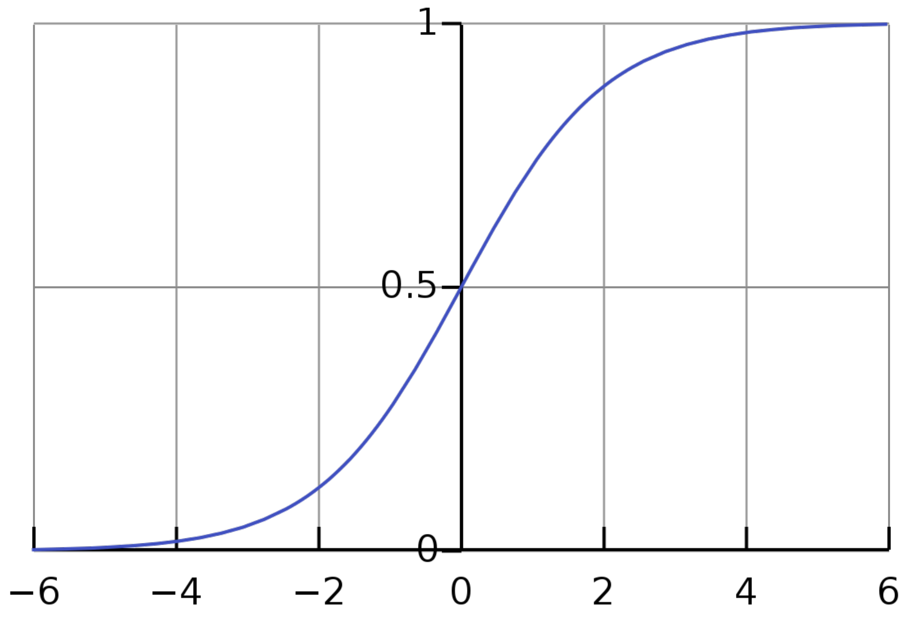

Figure 3.5: Plot of the sigmoid function [4].

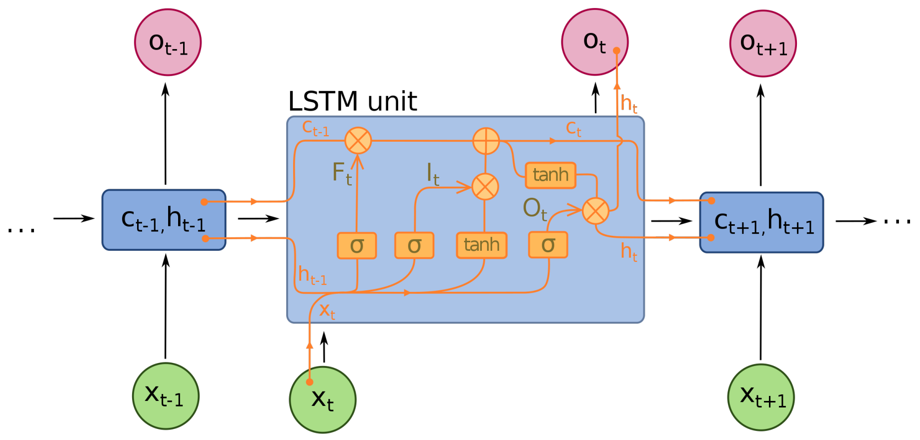

Figure 3.6: Example of RNN: Long term short memory cell [5].

from the sentence which are then fed into the fully connected network and classified as can be seen in image 3.7.

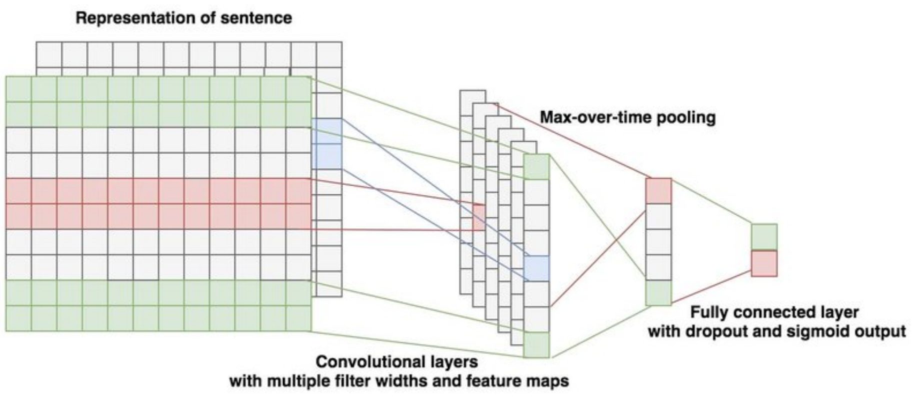
Representation of sentence
Max-over-time pooling
Fully connected layer
with dropout and sigmoid output
Convolutional layers
with multiple filter widths and feature maps

Figure 3.7: CNN for text classification [6].

3.3 The Age of the Transformer

Since the publication of Attention Is All You Need (Vaswani et. al., 2017) [7] many NLP researchers have been using the Transformer.

Attention

The key component of the Transformer is **attention**, which is a mechanism which allows the model to learn the parts of data which are more important than the rest. To put it simply it simulates how a human would pay attention to reading a text. [7]

There are multiple different flavors of attention. The most important concepts in the context of the Transformer are **key, query, value attention**, **self attention**, **multi-head attention** and **a scaled dot-product attention**. [7]

The key, query, value terms come from information retrieval. It represents mapping keys to some values and then querying those key when searching for values. In the Transformer case the embeddings matrix is multiplied by learned weight matrices W_k, W_q a W_v to create keys, queries and values matrices K, Q and V.

$$Z = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{3.6}$$

These matrices are then used to calculate Z using **the scaled dot product attention** formula from 3.6. The Z matrix is used to multiply the embeddings and produce the output.

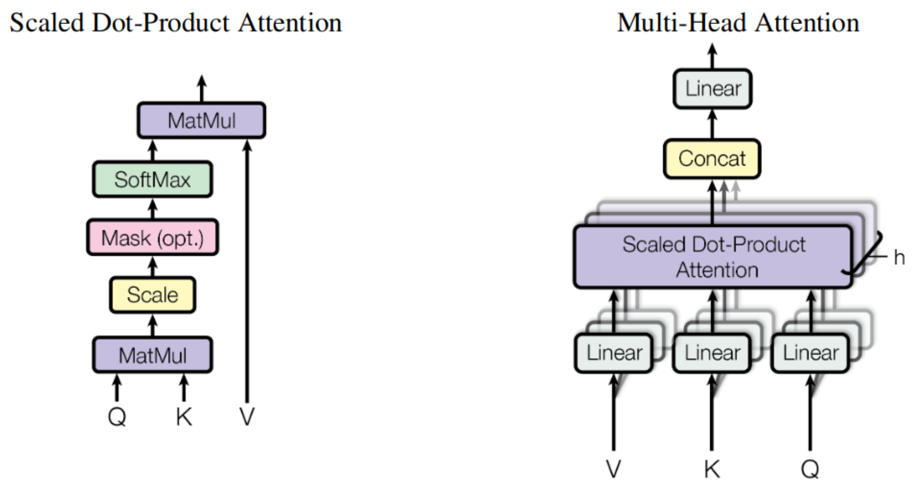

Figure 3.8: The attention mechanism [7].

In the actual transformer there are multiple heads performing **multi-head attention**. This simply means that each matrix W_{k_i}, W_{q_i} a W_{v_i} produces matrices K_i, Q_i and V_i resulting in Z_i for $i \in 0, ..., n$ where n is the number of attention heads. All the Z_i matrices are then concatenated and multiplied by a learned W_0 matrix into the final Z matrix as can be seen in 3.8.

The key, query, value attention can be calculated using embedding keys, queries and values from the same set which is called **self-attention** or using keys and values from one set and queries from another which is **cross-attention.** Both of these principles are used in different parts of the transformer.

The Transformer

The Transformer is a sequence to sequence model which compared to recurrent neural networks (RNNs) takes all of its input at the same time. Although this drastically improved its performance compared to strictly sequential RNNs, it poses a problem of determining the order of tokens in the input sequence.

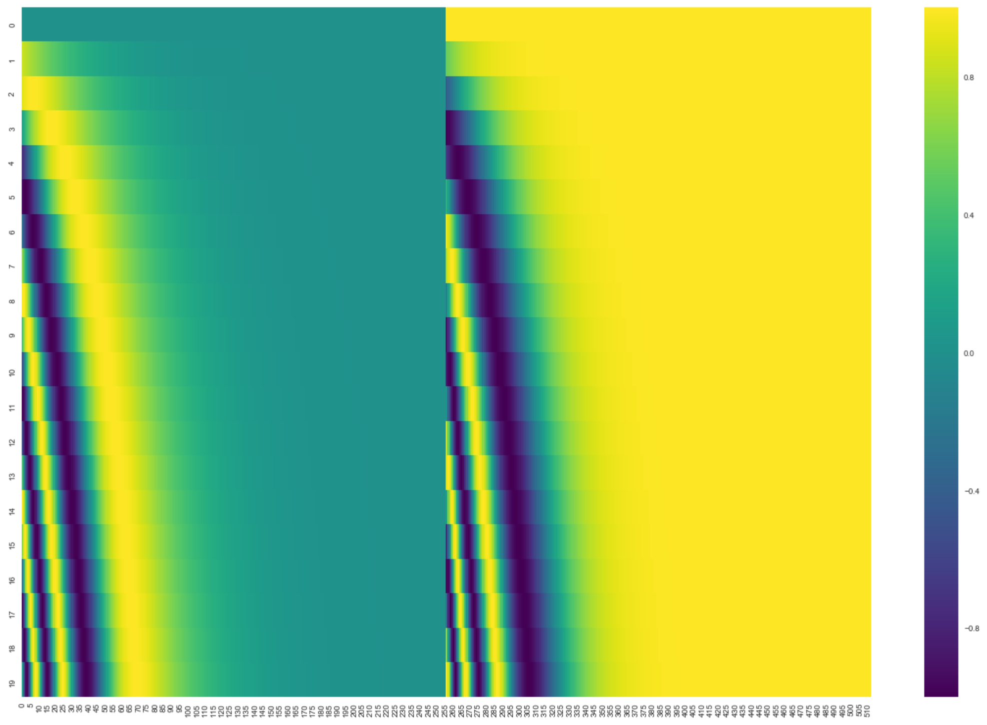

Figure 3.9: Positional encoding [7].

The solution is **positional encoding.** Each input embedding gets a positional vector added to it. These positional vectors can be learned or they can used a pre-calculated pattern. The pattern example can be seen in image 3.9.

The Transformer is based on the encoder-decoder architecture which can be seen in picture 3.10. Both encoder and decoder are composed of N stacked identical layers.

The encoder layer consists of a multi-head self-attention mechanism and a position-wise fully connected feed-forward network (FFN). There is a residual connection around each of these sub-layers, followed by layer-normalization. [7]

The decoder is similar to encoder in that it has also multi-head self-attention and position-wise feed-forward network as well as a residual connections and layer normalization. The difference is that the decoder adds a first sub-layer of multi-head cross-attention which performs the attention calcula-

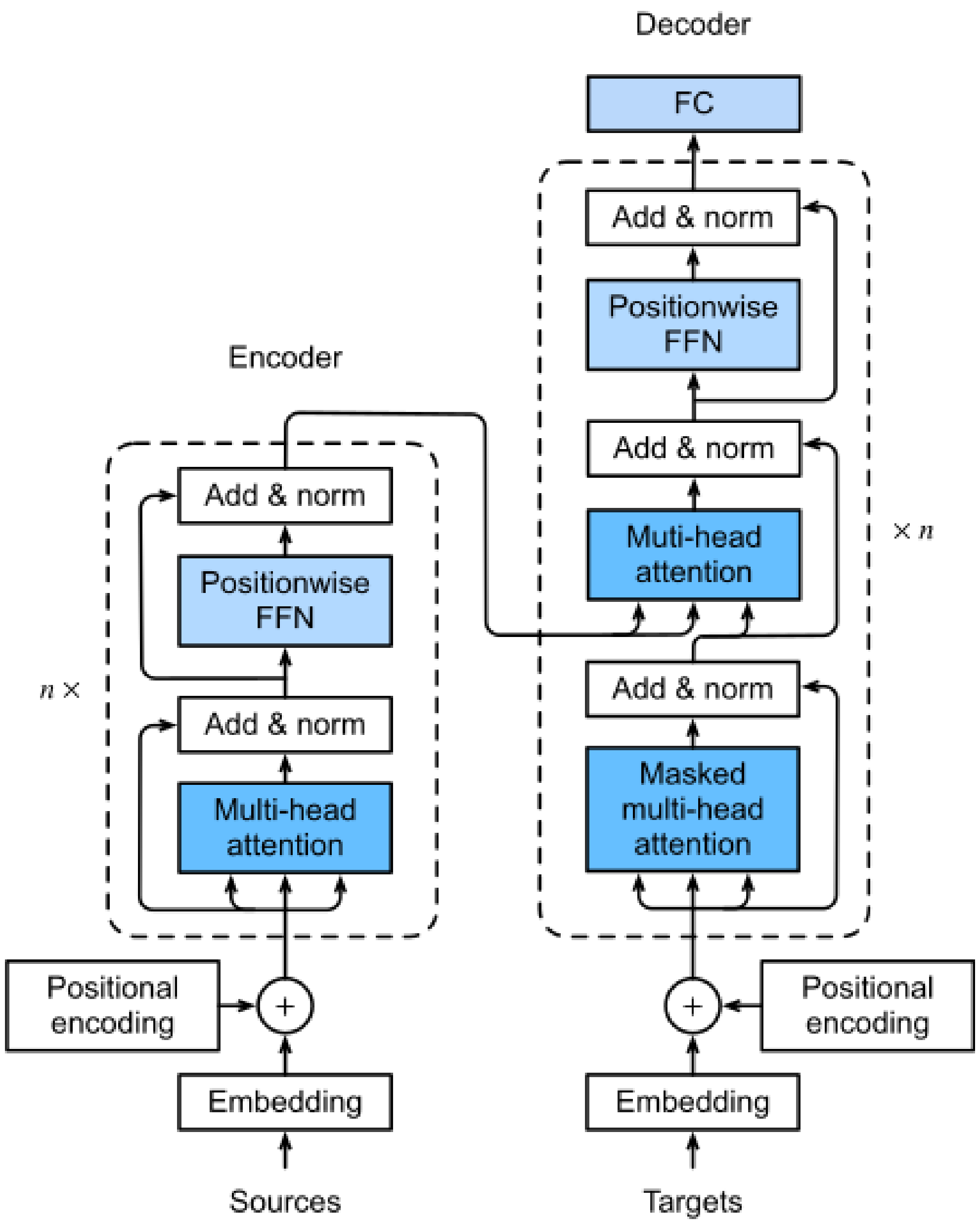

Figure 3.10: Transformer model architecture [7].

tion on the outputs of the encoder. Furthermore it uses masking to prevent positions from attending to subsequent positions. This combined with the output embedding being offset by one position, makes sure that predictions only depend on known outputs from previous positions. [7]

BERT

The transformer architecture inspired multiple pre-trained language models. BERT stands for Bidirectional Encoder Representations from Transformers. It uses up to 345 million parameters and supports more than 100 languages.

BERT was designed for fine-tuning, adding a custom output layer to the pre-trained network. *“...the pre-trained BERT representations can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications”* [8].

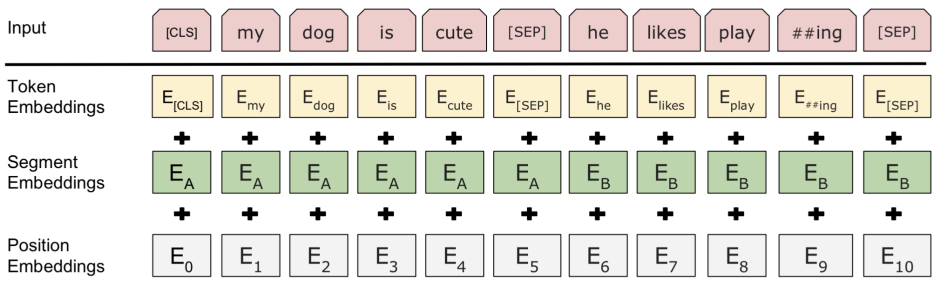
Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3.11: BERT input representation [8].

BERT outperformed previously used methods because it is the first unsupervised, deeply bidirectional system for pre-training NLP. BERT was trained using only an enormous plain text corpus publicly available on the web in many languages. [8]

3.4 Model Evaluation

When talking about machine learning models, it is important to define metrics for comparing models to each other. Since sentiment analysis is classification task, appropriate classification metrics have been chosen:

- Classification Accuracy,
- Precision,
- Recall,
- and F1 score.

Classification Accuracy

The accuracy of a classification model can be simply calculated as follows

$$\text{accuracy}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} 1(\hat{y}_i = y_i), \tag{3.7}$$

where y is a set of test samples, $\hat{y}$ a set of predictions and $1(x)$ an indicator function which is equal to 1 only if x is true. Clearly accuracy$(y, \hat{y}), \to [0, 1]$ where if it is 0 it means that no predictions were correct and if it's 1 then all predictions were correct.

Precision

In a simple binary classification, the precision metric is defined as follows

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{3.8}$$

where TP is the number of true positives and FP is the number of false positives.

A generalized version of precision for multiclass classification can be calculated as follows

$$\text{Precision} = \frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l| \, R(y_l, \hat{y}_l), \tag{3.9}$$

where L is the set of labels and $R(A, B) := \frac{|A \cap B|}{|A|}$.

Recall

In a simple binary classification, the recall metric is calculated as follows

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.10}$$

where TP is the number of true positives and FN is the number of false negatives.

A generalized version of recall for multiclass classification can be calculated as follows

$$\text{Recall} = \frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l| \, R(\hat{y}_l, y_l), \tag{3.11}$$

where L is the set of labels and $R(A, B) := \frac{|A \cap B|}{|A|}$.

F1 Score

The F-measure can be interpreted as a weighted harmonic mean of the precision and recall. A measure reaches its best value at 1 and its worst score at 0. In F1 score both recall and the precision are equally important. It is calculated as follows

$$\mathrm{f}_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{3.12}$$

A generalized version of the F1 score for multi-class classification can be calculated as follows

$$\mathrm{F}_1 = \frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l| \, \mathrm{f}_1(y_l, \hat{y}_l), \tag{3.13}$$

where $\mathrm{f}_1(A, B)$ is the binary formula from (3.12) applied for one class from L.

Chapter 4

Transfer Learning

Transfer learning will be the next driver of machine learning's commercial success after supervised learning. – Andrew Ng [28]

Traditional machine learning uses the training and testing data from the same input feature space and same data distribution. The is the fundamental presupposition for fitting statistical models.

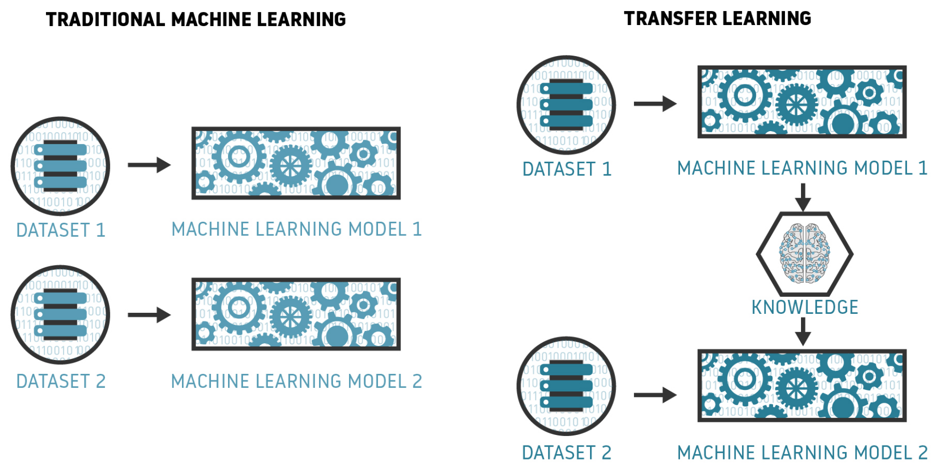

Figure 4.1: Illustration of transfer learning [9].

Formally, transfer learning (TL) is, according to Pan and Yang from 2009, defined as a domain D is a two-element tuple consisting of feature space $\mathcal{X}$ and marginal probability $\mathrm{P}\,X$, where X is a sample data point. The domain can be represented as $D = \{\mathcal{X}, P(X)\}$. [29]

For a given domain D a Task T is defined by two components $T = \{\mathcal{Y}, \mathrm{P}(Y|X)\}$ where $Y \in \mathcal{Y}$. $\mathcal{Y}$ is a label space and $\mathrm{P}(Y|X)\}$ is a predictive function learned from feature vector and label pairs (x_i, y_i) where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. [29]

Let D_S be a source domain, T_S be a corresponding source task and D_T and T_T target domain and task, the objective of transfer learning is to enable to learn the the target conditional probability distribution $\mathrm{P}(Y_T|X_T)$ in D_T using the information gained from D_S and T_S where $D_S \neq D_T$ or $T_S \neq T_T$. [29]

Parameter Transfer

Parameter-transfer method leverages the parameter sharing model of the source and target domains. Using the parameter transfer method, the trained model parameters can be transferred in a large number of datasets to the target task. [30]

An example of a parameter transfer method is Word2vec [2].

Instance Transfer

Instance-transfer method refers to the data sharing of the source and target domain. It can be filtered from the source domain by re-weighting. The data from the target domain can be augmented with the labeled source domain samples. [30]

An example of an instance transfer algorithm is TrAdaBoost which allows users utilize a small amount of newly labeled data and leverage old data to construct a classification model for new data. [31]

Feature Representation Transfer

Feature-representation-transfer method is based on the condition that the source domain and the target domain have part crossover features. It is necessary to transform the data of the two domains into the same feature space through feature transformation, and then perform traditional machine learning.

This method has lower requirements for similarity between the two domains, it is more widely used and performs well in various tasks of NLP. [30]

4.1 Transfer Learning in NLP

In some scenarios, including natural language processing, it might be quite difficult to obtain training data for a specific task (e. g. translation from a language with limited resources) or it might be computationally expensive to train a certain model (e. g. a large Transformer with many layers).

Transfer learning tries to solve these problems. It is a technique in machine learning which takes already learned knowledge from one problem and applies it to another. It simulates a phenomenon which occurs when humans go through the process of learning a new thing. For example, if someone knows how to play a guitar, it is going to be much easier to also learn how to play a piano. The basic knowledge of playing a musical instrument is already there giving them a head start before someone with no musical experience whatsoever.

This technique is particularly useful in domains such as image recognition. The hierarchical convolutional networks can be interpreted as identifying edges in the earlier layers, shapes in the middle layers and task-specific features in the later layers [32]. This allows to ”cut” away the last few layers of a fully trained network and keep a network capable of understanding shapes and simple objects.

The same can be performed for a natural language. There are a vast number of language tasks - machine translation, next sentence prediction, question answering, text classification and many more. All of those need to learn to understand the texts which are being fed into them. This is exactly where transfer learning comes into play. By removing the last few layers from an NLP model for translation, a basic model for understanding a particular language domain is left to be reused for another task.

Especially in NLP tasks transfer learning, which is more a general term, is sometimes called **domain adaptation** [33].

4.2 Related Work

DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

In 2020 Sanh et al. proposed a smaller and faster version of a BERT model [8] using a technique known as knowledge distillation [11]. BERT is in and of itself an example of transfer learning as it uses a large pre-trained model for fine-tuning to a specific task. DistilBERT attempts to get to a performance level of a regular BERT by highly optimizing its layers to fit into a parameter budget.

Simple, scalable adaptation for neural machine translation

Bapna et al. in 2019 [10] proposed a model architecture of light-weight adapters for neural machine translation. The details of this paper will be discussed in the next section as it is the main inspiration for this paper.

Multi-label aspect-based sentiment analysis using TL

In 2020 Tao et al. used a transfer learning extended aspect based sentiment analysis methods for multi-label classification. They also proposed an Aspect Enhanced Sentiment Analysis (AESA) for sequence classification with consideration of the entity aspects [34].

Deep transfer learning baselines for sentiment analysis in Russian

Smetanin et al. in 2021 fine-tuned Multilingual BERT, RuBERT, and two versions of the Multilingual Universal Sentence Encoder on seven sentiment analysis datasets [35]. They achieved state-of-the-art result using transfer learning on a low resource language like Russian.

4.3 Domain Specific Adapters

In 2019 a new approach was proposed in the field of neural machine translation. The motivation behind the idea by Bapna et al. was to create a simple, scalable architecture allowing to take advantage of high resource language knowledge in low resource language translation.

Their version is motivated by a 2019 paper Parameter-Efficient Transfer Learning for NLP by Houlsby et al. [33]. They proposed a new strategy for tuning a large model on several downstream tasks. They stated that this strategy has three key properties:

1. it attains good performance,

2. it does not require simultaneous access to all datasets and

3. it adds only a small number of extra parameters per task [33].

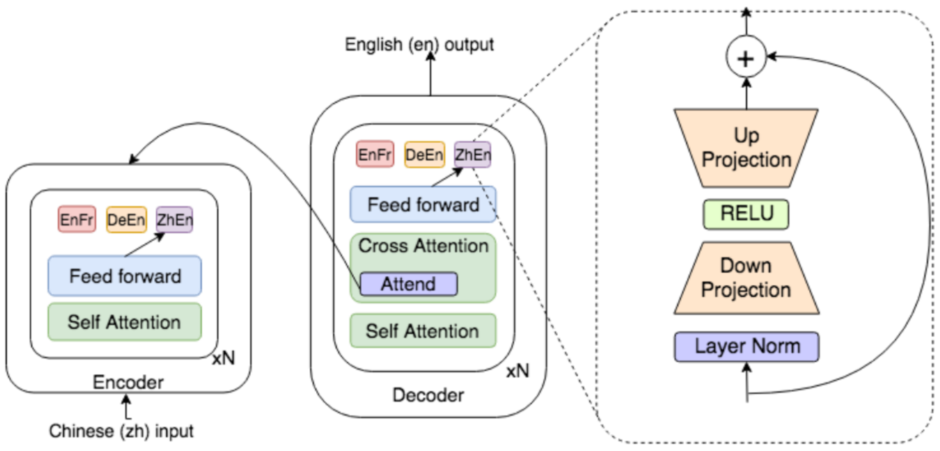

Figure 4.2: Adapter architecture [10].

Bapna et al.'s approach consisted of two phases – first train a generic base language model and second adapt it to new tasks using small network modules. They took a regular Transformer from [7] and pre-trained it on a large source corpus. After that the researchers added per-task light-weight adapters after every layer in both encoder and decoder blocks. Finally, they fine-tuned the parameters of these adapters on a specialized corpus. This algorithm can be performed for every task meaning that only one model has to trained from scratch and many tasks can be adapted on top of it. [10]

The adapter design had to be simple and flexible. The proposed architecture consists of layer normalization followed by a single layer feed forward down projection layer with a non-linear activation function. The projection dimension is the only hyper-parameter of the adapter making it very simple to tune. This embedding is then projected back up to its original size and the input vector is added to it. The residual connection is there to represent a *no operation* in case it is needed. The whole architecture can be seen in picture 4.2. [10]

In mathematical terms, let z_i be the output of the i-th layer with dimension d. Layer normalization LN is first applied to the input of the adapter for task T.

$$\tilde{z}_i^T = \underset{T}{\mathrm{LN}}(z_i). \tag{4.1}$$

Next the down projection onto the dimension b is applied. The dimension is picked based on the complexity of the task and size of the adaptation corpus.

$$h_i^T = \mathrm{ReLU}(W_{bd}^T \tilde{z}_i^T). \tag{4.2}$$

Finally the inner representation is up projected back to its original dimension d and the residual connection is added to it.

$$x_i^T = W_{db}^T h_i^T + z_i. \tag{4.3}$$

Chapter 5

Implementation

5.1 Datasets

The IMDb Dataset

The original movie reviews dataset contains 50000 highly polar movie reviews, 25000 for training and 25000 for testing [36]. This dataset is very popular in sentiment analysis tasks and all of NLP in general. For example in the past few years it has been used by Tripathy et al. to perform sentiment analysis using multiple different ML approaches [37], by Shaukat et al. who used both lexicon based methods as well as neural nets [38] and Ali at al. who used LSTMs and CNNs as well as hybrid CNN_LSTM to classify these review [39].

The IMDB dataset was used for its ubiquity which makes it a good benchmark for comparing new approaches with the rest of the world.

Dataset statistics

Statistic	**Value**
min length	4
max length	2470
mean length	231
median length	173
length standard deviation	171
10 most common words	['the', 'a', 'and', 'of', 'to', 'is', 'in', 'I', 'that', 'this']

Table 5.1: IMDb dataset statistics

Sample negative review:

```
Story of a man who has unnatural feelings for a pig. Starts out with a
     opening scene that is a terrific example of absurd comedy. A
    formal orchestra audience is turned into an insane, violent mob by
```

```
 the crazy chantings of it's singers. Unfortunately it stays
absurd the WHOLE time with no general narrative eventually making
it just too off putting. Even those from the era should be turned
off. The cryptic dialogue would make Shakespeare seem easy to a
third grader. On a technical level it's better than you might
think with some good cinematography by future great Vilmos
Zsigmond. Future stars Sally Kirkland and Frederic Forrest can be
seen briefly.
```

Sample positive review:

```
If you like adult comedy cartoons, like South Park, then this is
    nearly a similar format about the small adventures of three
    teenage girls at Bromwell High. Keisha, Natella and Latrina have
    given exploding sweets and behaved like bitches, I think Keisha is
     a good leader. There are also small stories going on with the
    teachers of the school. There's the idiotic principal, Mr. Bip,
    the nervous Maths teacher and many others. The cast is also
    fantastic, Lenny Henry's Gina Yashere, EastEnders Chrissie Watts,
    Tracy-Ann Oberman, Smack The Pony's Doon Mackichan, Dead Ringers'
    Mark Perry and Blunder's Nina Conti. I didn't know this came from
    Canada, but it is very good. Very good!
```

The Skytrax Airline Reviews Dataset

Skytrax is consultancy company in the United Kingdom specializing in airline and airport reviews [40].

The Skytrax dataset is a collection of more than 64000 airline reviews given by customers via a questionnaire [41]. The data from years between 2006 and 2019 includes 1-5 and 1-10 point reviews as well as free text answers making it a great choice for classification. The 1-10 scales can be simply transformed into many n-class classification tasks.

The dataset is enormous, making it suitable for pretraining models to be later used for fine-tuning and/or adapting. The dataset is skewed towards positive reviews (6-10 points) - around 53 % to the 47 % of negative reviews (1-5 points) but it is still pretty balanced.

The Skytrax dataset has been used by Hui et al. in 2019 to create an explainable NLP system [42].

To conclude, this dataset was picked for its size and being of a different domain to movie reviews.

Dataset Statistics

Sample negative review:

```
Trip Verified | Istanbul to Bucharest. We make our check in in the
    airport, they Take our luggage , we go to the gate and at the gate
     surprise they dont let you board with two children, because they
```

Statistic	**Value**
min length	1
max length	1003
mean length	138
median length	112
length standard deviation	94
10 most common words	['the', 'to', 'and', 'was', 'a', 'I', 'in', 'of', 'on', 'for']

Table 5.2: Skytrax dataset statistics

```
    say the flight is overbooked. We had to wait in the airport with
    two children until 5 oclock in the morning until they bring you to
     a hotel 2 hours far away from the airport without luggage,
    without eat without nothing. Our first and last flight with this
    airline.
```

Sample positive review:

```
Trip Verified | Flew on Turkish Airlines IAD-IST-KHI and return KHI-
    IST-IAD. Turkish Airlines has consistently maintained its quality
    since I first flew with them in 2007. The flights leave on time,
    the catering is excellent, the inflight entertainment is extensive
     and the interface easy to use, and the cabin crew is excellent.
    Interesting though the A330 on the KHI-IST route and return seemed
     to have more leg room and was newer than the A330 on the IAD-IST
    route which was showing its age. The A330 on the IAD-IST route had
     a slow responding interface for the inflight entertainment and a
    broken table on the return flight. But Turkish Airlines will be
    replacing the A330 on its flight to IAD with the 787 sometime in
    the summer. Turkish food was served on the return leg which I
    personally like, and I saw the cabin staff helping elderly
    passengers walk to the lavatory which was nice. Overall another
    wonderful experience with Turkish Airlines."
```

The Airline Tweets Dataset

Twitter is a very popular platform for people to voice their opinions about their customer experiences [43]. The dataset of tweets is smaller than the other two. It consists of 14000 English tweets describing people's experiences with airlines from February 2015. The opinions are highly skewed towards the negative with 63 % to 21 % neutral and only 16 % positive.

The tweets have been also widely used in the NLP field, e. g. Rustam et al. used multiple text representations and ML methods to classify it [44], Wan et al. tried six different classification approaches including ensemble methods to improve maximize the accuracy [45] and Kumar et al. extracted its features using word embeddings with Glove dictionary approach and n-gram approach for SVM and multiple NN architectures [46].

The tweets dataset shares the same domain (airline reviews) with the Skytrax dataset making it a good choice for comparing in domain knowledge transfer.

Dataset statistics

Statistic	Value
min length	2
max length	36
mean length	19
median length	20
length standard deviation	6.7
10 most common words	['to', 'the', 'I', 'a', 'for', 'and', '@united', 'on', 'you', 'my']

Table 5.3: Tweets dataset statistics

Sample negative review:

```
@AmericanAir you have my money, you change my flight, and don't answer
     your phones! Any other suggestions so I can make my commitment??"
```

Sample positive review:

```
@VirginAmerica not worried, it's been a great ride in a new plane with
     great crew. All airlines should be like this."
```

5.2 Tools

Python

Python3 is a general purpose programming language created by Guido van Rossum in 1991. It is the most popular among data scientists and researchers in general for its low barrier of entry, easy syntax and many great packages for working with data. [47]

A package is a self-contained code library dealing with a specified task, e.g. working with tables or machine learning algorithms. As the number one choice for data scientists, there are many packages solving everyday tasks in machine learning research while abstracting complicated implementation away.

The most useful Python packages used during the implementation phase of this thesis are Numpy, Pandas and Scikit learn.

Numpy implements data types and methods to work with multi-dimensional numerical data. It is implemented in the C language making it much faster to work with high volume data than Python. [48]

Pandas is a data analysis and manipulation tool for Python. It is mostly used in the data exploration and preprocessing phase of any machine learning task. [49]

Scikit learn is a machine learning library offering not only ready to use statistical models but also preprocessing, evaluation and data augmentation tools. This thesis mostly uses it for data splitting. [50]

TensorFlow + Keras

TensorFlow is a free and open-source library developed by Google Brain team in 2015. It is one of the most used libraries in the field of training and inference of deep neural networks. The TensorFlow backend is implemented and optimized for multiple platforms such as CPUs, GPUs and TPUs. This allows it take advantage of all the performance available by the hardware. [51]

TensorFlow supports a number of APIs into multiple programming languages, most notably Python, C++ and JavaScript. Keras is a very popular high-level API for Python. High-level means that it is easier to create simple working prototypes without the need to build custom components and configuring a lot of parameters. [51]

When building a model in Keras, **layers** are combined using either a sequential or functional approach. A sequential model allows to append layers to a model until the desired model is built. The functional approach uses recursion to send outputs of one layer into another layer. It makes building some more complex architectures very elegant. Therefore, it was the approach of choice in this thesis. [52]

Google Colaboratory

Google Colaboratory is cloud notebook environment. Google offers CPU, GPU and TPU hardware to computationally intensive models for free. In addition to that having the source code in the cloud makes it safe and easy to share among other researchers. Since NLP models perform well on GPUs, Google Colaboratory was an obvious choice for this thesis. [53]

5.3 Model

The model is based on the Transformer mentioned in section 3.3. The actual implementation is heavily inspired by the Keras code example Text classification with Transformer [54].

The implementation consists of three parts:

1. token and positional embedding,
2. the adapter
3. and the actual transformer block.

Token and Positional Embedding

The token and positional embedding takes maximum length of a sequence *maxlen*, the size of the corpus *vocab_size* and the desired dimension of an embedding *embed_dim* as arguments.

This layer utilizes two Embedding layers from Keras [55]. The first takes the input and creates a token embedding of size *embed_dim*. The second takes the size of the input and adds an embedding of an increasing vector $[0, 1, 2, ..., maxlen]$ to the token embedding. The sum of these two embeddings is the output of this layer.

```
from tensorflow.keras import layers

class TokenAndPositionEmbedding(layers.Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super(TokenAndPositionEmbedding, self).__init__()
        self.token_emb = layers.Embedding(input_dim=vocab_size,
            output_dim=embed_dim)
        self.pos_emb = layers.Embedding(input_dim=maxlen,
            output_dim=embed_dim)

    def call(self, x):
        maxlen = tf.shape(x)[-1]
        positions = tf.range(start=0, limit=maxlen, delta=1)
        positions = self.pos_emb(positions)
        x = self.token_emb(x)
        return x + positions
```

Figure 5.1: Token and positioning Embedding code

Adapter

The adapter has been implemented based on the adapter from Simple, Scalable Adaptation for Neural Machine Translation from section 4.3.

Its only argument is the desired embedding dimension and it is therefore its only hyperparameter.

The adapter consists of five layers:

1. initial dropout,
2. layer normalization,
3. dense layer with half of the embedding dimension and ReLU activation,
4. dense layer with linear activation and
5. final dropout.

Finally, when the input gets sequentially modified throughout the adapter, it is added to the original output to allow the adapter to represent a no-op.

```
from tensorflow.keras import layers

class Adapter(layers.Layer):
    def __init__(self, embed_dim, rate=0.1):
        super(Adapter, self).__init__()
        self.layernorm = layers.LayerNormalization(epsilon=1e-6)
        self.relu = layers.Dense(embed_dim / 2, activation="relu",
            name='ReLU')
        self.linear = layers.Dense(embed_dim, activation="linear",
            name='Up_projection')
        self.dropout1 = layers.Dropout(rate)
        self.dropout2 = layers.Dropout(rate)
        self.add = layers.Add()

    def call(self, inputs):
        x = self.dropout1(inputs)
        x = self.layernorm(x)
        x = self.relu(x)
        x = self.linear(x)
        x = self.dropout2(x)
        return self.add([inputs, x])
```

Figure 5.2: Adapter code

Transformer block

The transformer block combines all the previous parts together and forms the main building block of the final model.

It is actually just the transformer encoder as the decoder is not necessary for classification purposes. The block consists of seven layers in total:

1. the Keras MultiHeadAttention layer, which implements the multihead attention algorithm mentioned in subsection 3.3,
2. first dropout layer,
3. first layer normalization,
4. a two layer feed forward network with ReLU activation in the middle,
5. second dropout layer,
6. second layer normalization and
7. lastly, the adapter 5.2 mentioned in the previous paragraph.

```
from tensorflow.keras import layers

class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = layers.MultiHeadAttention(num_heads=num_heads,
            key_dim=embed_dim)
        self.ffn = keras.Sequential(
            [layers.Dense(ff_dim, activation="relu"), layers.Dense(
                embed_dim),]
        )
        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = layers.Dropout(rate)
        self.dropout2 = layers.Dropout(rate)
        self.adapter = Adapter(embed_dim)

    def make_only_adapter_trainable(self):
        self.att.trainable = False
        self.ffn.trainable = False
        self.layernorm1.trainable = False
        self.layernorm2.trainable = False
        self.dropout1.trainable = False
        self.dropout2.trainable = False

    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        layer_norm2 = self.layernorm2(out1 + ffn_output)
        return self.adapter(layer_norm2)
```

Figure 5.3: Transformer block code

Complete Model

The last part necessary for a classification model is the output layer. In this case of sequence classification, after the final transformer block a *GlobalAveragePooling1D* makes an average of all the encoded tokens from a sequence. This averaged embedding is than run through one dropout layer, a dense layer of exactly 20 neurons with ReLU as its activation function, another dropout and its output using a one neuron with a sigmoid activation function.

The final model consisting of the token and positional embedding layer 5.1, n transformer blocks 5.3 with adapters 5.2 and the final classification output layer.

```
Model: "imdb_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 200)]             0

 token_and_position_embeddin (None, 200, 64)           1292800
 g (TokenAndPositionEmbeddin
 g)

 transformer_block (Transfor (None, 200, 64)           62688
 merBlock)

 transformer_block_1 (Transf (None, 200, 64)           62688
 ormerBlock)

 global_average_pooling1d (G (None, 64)                0
 lobalAveragePooling1D)

 dropout_8 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 20)                1300

 dropout_9 (Dropout)         (None, 20)                0

 dense_5 (Dense)             (None, 1)                 21

=================================================================
Total params: 1,419,497
Trainable params: 1,419,497
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.4: Example classification model summary

The example of a compiled model using all the previously mentioned parts and the hyperparameters of $embed_dim = 64$ and $num_blocks = 2$ can be seen in figure 5.4.

Chapter 6

Methodology

The experiment is a **document level binary classification** of three datasets – IMDb, Skytrax and Airline tweets using a **machine learning** approach. There are three scenarios which are being explored:

1. base models only,
2. usage of a base model with adapters and
3. usage of a base model with fine-tuning.

All the performances of these scenarios are later compared to each other and to a third party benchmark.

6.1 Data Split

All of the following experiments have been performed on the same exact data. The IMDb dataset comes split half and half - training data and testing data. Furthermore, the training data has been split into training and validation data using the ratio of 0.2. The number of positive reviews is the same as the number of negative ones making the dataset perfectly balanced.

The absolute numbers can be seen in table 6.1.

The Skytrax dataset was split 30 % test data and validation data 20 % of the rest leaving 56 % of total training data.

The absolute numbers can be seen in table 6.2.

The Tweets dataset, even though it is smaller, was split 30 % test data and validation data 20 % of the rest leaving 56 % of total training data.

As can be seen in table 6.3, the tweets dataset is highly skewed towards the positive labels, therefore, a class weighting compensation has been used when training the models.

Dataset subset	**Percentage**	**Count**
Training data	40 %	20000
Testing data	50 %	25000
Validation data	10 %	5000
Positive training data	50 %	10000
Negative training data	50 %	10000
Positive validation data	50 %	2500
Negative validation data	50 %	2500
Positive testing data	50 %	12500
Negative testing data	50 %	12500

Table 6.1: Train, test, validation split of the IMDb dataset

Dataset subset	**Percentage**	**Count**
Training data	56 %	35848
Validation data	14 %	8963
Testing data	30 %	19206
Positive training data	52 %	18932
Negative training data	48 %	16916
Positive validation data	53 %	4823
Negative validation data	47 %	4140
Positive testing data	53 %	10246
Negative testing data	47 %	8960

Table 6.2: Train, test, validation split of the Skytrax dataset

6.2 Preprocessing

The raw textual data are not immediately usable for training and as such need to be preprocessed. The algorithm of preprocessing chosen for these experiments is follows established steps. The data from all three datasets undergoes these standardization steps:

- lower-casing,
- removal of HTML tags and
- removal of punctuation.

The removal of stop words was also considered and experimented with. Nevertheless, it turned out to negatively effect the performance there it has not been used.

After the data has been standardized it is converted to a vectorized form using the Keras *TextVectorization* layer. [52]

Dataset subset	**Percentage**	**Count**
Training data	56 %	6462
Validation data	14 %	1616
Testing data	30 %	3463
Positive training data	79 %	5108
Negative training data	20 %	1354
Positive validation data	80 %	1299
Negative validation data	20 %	317
Positive testing data	80 %	2771
Negative testing data	20 %	692

Table 6.3: Train, test, validation split of the Tweets dataset

6.3 Scenarios

The scenarios selected for the experiments part of this thesis are as follows:

1. fitting a model with adapters for each of the three datasets,
2. fitting a base model with adapters on the IMDb dataset, evaluating on the other datasets before and after fitting the adapters with those datasets,
3. fitting a base model with adapter on the Skytrax dataset, evaluating on the other datasets before and after fitting the adapters with those datasets,
4. fitting a base model with adapters on the IMDb dataset, evaluating on the other datasets before and fine-tuning the whole model with those datasets and
5. fitting a base model with adapters on the Skytrax dataset, evaluating on the other datasets before and fine-tuning the whole model with those datasets.

Base Model

The first scenario establishes the performance of all models on all the datasets without the use of any transfer learning techniques. It should serve as a baseline for comparing the results from the advanced scenarios.

Using Adapters

The second scenario is designed to show how fitting the adapters increases the performance of the base model on different out of domain datasets. Similarly, the third scenario is designed to do the same on in domain datasets of Skytrax and airline tweets.

In these scenarios the flow of the experiment goes as follows:

1. build the base model as it is shown in 5.4,
2. fit the base model using the base dataset (IMDb, Skytrax),
3. evaluate the base model using the test data on that dataset,
4. evaluate the base model using the test data on the secondary dataset,
5. **make all parameters non-trainable except for the adapters**,
6. fit the secondary dataset,
7. evaluate the model on that dataset,
8. evaluate the model on the Tweets dataset,
9. fit the Tweets dataset and
10. finally evaluate the model on the Tweets dataset again.

Fine-tuning

The fourth and fifth scenarios are the traditional fine-tuning approaches used to compare the fitting of the light-weight adapters to the fitting of the entire network of the IMDb base model and Skytrax base model respectively.

In these scenarios the flow of the experiment goes as follows:

1. build the base model as it is shown in 5.4,
2. fit the base model using the base dataset (IMDb, Skytrax),
3. evaluate the base model using the test data on that dataset,
4. evaluate the base model using the test data on the secondary dataset,
5. **leave all parameters trainable**,
6. fit the secondary dataset,
7. evaluate the model on that dataset,
8. evaluate the model on the Tweets dataset,
9. fit the Tweets dataset and
10. finally evaluate the model on the Tweets dataset again.

6.4 Hyperparameters

The hyperparameters used in all of the experiments are as follows:

- the dimension of the transformer embedding - *embed_dim*,
- the number of attention head - *num_heads*,
- the dimension of the feed forward network inside the transformer - *ff_dim*,
- number of transformer blocks - *num_transformer_blocks*,
- the size of the batch - *batch_size*,
- number of epochs - *epochs* and
- maximum length of a sequence - *sequence_length*.

Hyperparameter	**Value**
embed_dim	64
num_heads	3
ff_dim	32
num_transformer_blocks	2
batch_size	32
epochs	20
sequence_length	200

Table 6.4: Hyperparameter values for all experiments

The choices of hyperparameters which can be seen in table 6.4 produce a **base model** with around *1.4 million* parameters of which around *8.6 thousand* are the parameters of the **adapters**.

6.5 Benchmark

Model	**Test accuracy**	**Params (Millions)**
BERT base	0.93	110
DistilBERT	0.92	66

Table 6.5: Performance and size of BERT-base and DistilBERT on the IMDb dataset [11].

As a benchmark the 2019 paper DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter by Sanh et al. mentioned in section 4.2

was used [11]. This paper was chosen because of the researchers' intentions (reducing model size while keeping performance) being the same to those of this thesis, moreover because of their choice of datasets. They used a non-augmented IMDb dataset to fit a BERT-base from [8] and their proposed DistilBERT model. The accuracy on the test dataset together with the model sizes can be seen in table 6.5.

Chapter 7

Results

7.1 Base Models

Dataset	Accuracy	Precision	Recall	F1
imdb base train	1.0	1.0	1.0	1.0
imdb base val	0.86	0.84	0.89	0.86
imdb base test	0.84	0.84	0.84	0.83
skytrax base train	0.96	0.98	0.95	0.96
skytrax base val	0.9	0.93	0.89	0.91
skytrax base test	0.9	0.92	0.88	0.9
tweets train	0.98	0.99	0.93	0.96
tweets val	0.89	0.73	0.75	0.73
tweets test	0.91	0.76	0.76	0.74

Table 7.1: Results for base models only.

In the base model scenario without the usage of any transfer learning techniques the models performance can be found in table 7.1.

The IMDb model reached a perfect performance of all metrics equal to one in the training phase. On the validation dataset it reached around 0.85–0.89 in all the metrics and on the testing data the metrics were all around 0.84.

The Skytrax model got very close to a perfect model with scores around 0.97 on the training dataset. The score of both validation and test data were very close with both accuracy and F1 score being 0.9.

The Tweets dataset model also reached an almost perfect model with metrics close to 1 on the training dataset. The performance of the validation and testing data was around 0.9 accuracy and the other metrics in the vicinity of 0.75.

7.2 IMDb Base Model

The metrics in the section are from the experiment where the IMDb dataset was used to train the base model, then the Skytrax dataset was used to either adapt or fine-tune the model and finally the model was fitted on the Tweets dataset.

IMDb Adapters

Dataset	**Accuracy**	**Precision**	**Recall**	**F1**
imdb base train	1.0	1.0	1.0	1.0
imdb base val	0.86	0.84	0.89	0.86
imdb base test	0.84	0.84	0.84	0.83
skytrax test on imdb base	0.72	0.76	0.69	0.72
tweets test on imdb base	0.61	0.31	0.71	0.41
skytrax adapters train	0.99	1.0	0.98	0.99
skytrax adapters val	0.88	0.92	0.86	0.89
skytrax adapters test	0.88	0.91	0.86	0.88
tweets test on skytrax adapters	0.78	0.47	0.8	0.58
tweets adapters train	0.98	0.9	0.99	0.94
tweets adapters val	0.9	0.69	0.85	0.75
tweets adapters test	0.89	0.68	0.77	0.71

Table 7.2: Results for the IMDb base model with Adapters

In the table 7.2, the results from the IMDb base model with adapters can be seen. In this particular case, the training metrics on the base dataset indicate it was a perfect model with accuracy, precision, recall and F1 score all equal to 1. The validation and test metrics are all around 0.84-0.86.

On the Skytrax dataset before fitting the adapters of the base model showed both accuracy and F1 score of 0.72. After training **the adapters** on this dataset the performance in both previously mentioned metrics on the test dataset went up to 0.88.

The Tweets dataset's performance was low by having a recall of 0.31 on the IMDb base model which brought the F1 score to 0.41. After training the Skytrax adapters the performance on Tweets rose up to accuracy of 0.78 and but recall and F1 remained low around 0.47 and 0.58 respectively. When **the adapters** were trained, all the metrics went up, e.g accuracy to 0.89 and F1 score to 0.71.

Dataset	Accuracy	Precision	Recall	F1
imdb base train	1.0	1.0	1.0	1.0
imdb base val	0.86	0.84	0.89	0.86
imdb base test	0.84	0.84	0.84	0.83
skytrax test on imdb base	0.72	0.76	0.69	0.72
tweets test on imdb base	0.61	0.31	0.71	0.41
skytrax fine-tuned train	0.99	0.99	0.98	0.99
skytrax fine-tuned val	0.89	0.91	0.88	0.89
skytrax fine-tuned test	0.89	0.91	0.88	0.89
tweets test on skytrax fine-tuned	0.76	0.45	0.78	0.56
tweets fine-tuned train	0.98	0.91	1.0	0.95
tweets fine-tuned val	0.89	0.67	0.85	0.74
tweets fine-tuned test	0.88	0.68	0.79	0.72

Table 7.3: Results for the IMDb base model using Fine tuning

IMDb Fine-tuning

In the fine-tuning version of this experiment, which can be seen in table 7.3, the IMDb base model remained the same with perfect metrics on the training dataset and tightly around 0.85 in all the metrics on the test dataset.

The Skytrax dataset reached accuracy of 0.72 and F1 score of 0.71. After the whole model was **fine-tuned** using the Skytrax dataset, test accuracy as well as F1 score reached 0.89.

The Tweets dataset achieved low accuracy of 0.61 on the IMDb base model and reached accuracy of 0.67 on the Skytrax fine-tuned model and with recall at 0.31 the F1 score got to only 0.41. After fine-tuning it with the last dataset the performance reached accuracy of 0.88 and F1 score of 0.72.

7.3 Skytrax Base Model

The metrics in the section are from the experiment where the Skytrax dataset was used to train the base model, then the IMDb dataset was used to either adapt or fine-tune the model and finally the model was fitted on the Tweets dataset.

Skytrax Adapters

In the table 7.4, the results from the Skytrax base model with adapters can be seen. In this particular case, the training metrics on the base dataset indicate it was very close to a perfect model with accuracy, precision, recall and F1 score all around 0.96. The validation and test accuracy are both 0.9 with the rest of the metrics all around 0.9.

Dataset	**Accuracy**	**Precision**	**Recall**	**F1**
skytrax base train	0.96	0.98	0.95	0.96
skytrax base val	0.9	0.93	0.89	0.91
skytrax base test	0.9	0.92	0.88	0.9
imdb test on skytrax base	0.68	0.67	0.71	0.68
tweets test on skytrax base	0.79	0.47	0.77	0.57
imdb adapters train	0.87	0.82	0.95	0.87
imdb adapters val	0.83	0.78	0.91	0.84
imdb adapters test	0.83	0.78	0.91	0.84
tweets test on imdb adapters	0.8	0.49	0.77	0.58
tweets adapters train	0.97	0.9	1.0	0.94
tweets adapters val	0.91	0.72	0.89	0.78
tweets adapters test	0.88	0.66	0.83	0.72

Table 7.4: Results for the Skytrax base model with Adapters

On the IMDb dataset before fitting the adapters of the base model showed both accuracy and F1 score of 0.68. After training **the adapters** the performance in both previously mentioned metrics on the test dataset went up to 0.83 and 0.84 respectively.

On the Tweets dataset the performance on the Skytrax base model was low by having a recall of 0.47 on the base model which brought the F1 score to 0.54. After the IMDb adapter were trained, the performance rose to accuracy of 0.8 but the F1 score stayed low at 0.57. When **the adapters** were trained on the Tweets dataset, all the metrics went up, e.g accuracy to 0.89 and F1 score to 0.71.

Skytrax Fine-tuning

In the fine-tuning version of this experiment which can be seen in table 7.5, the Skytrax base model remained the same with almost perfect metrics on the training dataset and tightly around 0.9 in all the metrics on the validation and test dataset.

The IMDb test dataset reached accuracy as well as the F1 score of 0.84 after the the whole model was fine-tuned, the exact same as in the adapter scenario. Before that it only had accuracy and F1 of 0.68 on the Skytrax base.

The Tweets dataset reached accuracy of 0.73 on the Skytrax base model and with recall at 0.47 the F1 score got to only 0.57. After fine-tuning it with IMDb it did not get better arguably getting worse in F1 score of 0.52. Nevertheless after fine-tuning the already IMDb fine-tuned model with the last dataset the performance reached accuracy of 0.89 and F1 score of 0.71.

Dataset	**Accuracy**	**Precision**	**Recall**	**F1**
skytrax base train	0.96	0.98	0.95	0.96
skytrax base val	0.9	0.93	0.89	0.91
skytrax base test	0.9	0.92	0.88	0.9
imdb test on skytrax base	0.68	0.67	0.71	0.68
tweets test on skytrax base	0.79	0.47	0.77	0.57
imdb fine-tuned train	0.89	0.86	0.94	0.89
imdb fine-tuned val	0.83	0.8	0.88	0.84
imdb fine-tuned test	0.84	0.81	0.89	0.84
tweets test on imdb fine-tuned	0.7	0.39	0.87	0.52
tweets fine-tuned train	0.97	0.87	1.0	0.92
tweets fine-tuned val	0.9	0.69	0.91	0.77
tweets fine-tuned test	0.87	0.63	0.84	0.71

Table 7.5: Results for the Skytrax base model using Fine tuning

Chapter 8

Discussion

In the following sections, the results of base models are compared to dataset cross model evaluation where datasets were evaluated before the base models were trained on that dataset and to actual fine-tuned or adapted models.

8.1 Fine-tuning vs Base Models

Model	Dataset	Accuracy	Precision	Recall	F1
base imdb	imdb test	**0.84**	0.83	0.84	0.83
base skytrax	skytrax test	**0.9**	0.92	0.88	0.9
base tweets	tweets test	**0.91**	0.76	0.76	0.74
base imdb	skytrax test	0.72	0.76	0.69	0.72
base imdb	tweets test	0.61	0.31	0.71	0.41
base skytrax	imdb test	0.68	0.67	0.71	0.68
base skytrax	tweets test	0.79	0.47	0.77	0.57
ISF	skytrax test	0.89	0.91	0.88	0.89
ITF	tweets test	0.88	0.68	0.79	0.72
SIF	imdb test	**0.84**	0.81	0.89	0.84
STF	tweets test	0.87	0.63	0.84	0.71

Table 8.1: Comparison of test results from base models, pre-fine-tuning and fine-tuning. ISF = IMDb base model with Skytrax fine-tuning, ITF = IMDb base model with Tweets fine-tuning, SIF = Skytrax base model with IMDb fine-tuning and STF = Skytrax base model with Tweets fine-tuning.

In the table 8.1, the performance of base models compared to the fine-tuned models can be seen. The IMDb base model and the SIF (Skytrax base with IMDb fine-tuning) both reached accuracy of 0.84 therefore both of these approaches are equivalent.

The Skytrax base model reached 0.9 in all the metrics while ISF (IMDb base model with Skytrax fine-tuning) performed a bit worse but very close with accuracy and F1 score of 0.89. As a result both of these approaches are also equivalent.

On the Tweets dataset the performances of fine-tuned did not reach the highs of the base model of accuracy 0.91 and F1 of 0.74. The ITF (IMDb base model with Tweets fine-tuning) got the closest with accuracy 0.88 and F1 of 0.72. The in-domain similarity of Skytrax and Tweets can be observed in the Skytrax base on Tweets evaluation where the model without ever seeing the data reached 0.79 accuracy which with IMDb base it reached only 0.61.

8.2 Adapters vs Base Models

Model	Dataset	Accuracy	Precision	Recall	F1
base imdb	imdb test	**0.84**	0.83	0.85	**0.84**
base skytrax	skytrax test	**0.9**	0.9	0.9	**0.9**
base tweets	tweets test	**0.91**	0.76	0.76	**0.74**
base imdb	skytrax test	0.72	0.76	0.69	0.72
base imdb	tweets test	0.61	0.31	0.71	0.41
base skytrax	imdb test	0.68	0.67	0.71	0.68
base skytrax	tweets test	0.79	0.47	0.77	0.57
ISA	skytrax test	0.88	0.91	0.86	0.88
ITA	tweets test	0.89	0.68	0.77	0.71
SIA	imdb test	0.83	0.78	0.91	0.84
STA	tweets test	0.88	0.66	0.83	0.72

Table 8.2: Comparison of test results from base models, pre-adapters and adapted. Boldface indicates the best accuracy and F1 score for each dataset. ISA = IMDb base with Skytrax adapters, ITA = IMDb base with Tweets adapters, SIA = Skytrax base with IMDb adapters and STA = Skytrax base with Tweets adapters.

The performance of models with adapters can be seen in table 8.2. The performance of the adapter variants compared to the base models is lower nonetheless it is comparable. The base IMDb model reaches accuracy of 0.84 while the SIA (Skytrax base adapted on the IMDb dataset) reached 0.83 which is almost exactly the same performance.

The same can be said about the Skytrax base having 0.9 accuracy and the ISA (IMDb base adapted on the Skytrax dataset) reaching 0.88 making them comparable in performance.

Performance on the Tweets dataset was low before the adapting with accuracy of 0.61 and F1 of 0.41 on the IMDb base and 0.73 and 0.54 on the

Skytrax base. After the model was adapted these numbers rose up to accuracy of 0.89 from the IMDb base and 0.88 from the Skytrax base making it very similar to the Tweets base model with accuracy of 0.91.

8.3 Adapters vs Fine-tuning

Model	Params	Dataset	Accuracy	Precision	Recall	F1
ISF	1400	skytrax test	**0.89**	0.91	0.88	**0.89**
ITF	1400	tweets test	0.88	0.68	0.79	**0.72**
SIF	1400	imdb test	**0.84**	0.81	0.89	**0.84**
STF	1400	tweets test	0.87	0.63	0.84	0.71
ISA	8.6	skytrax test	0.88	0.91	0.86	0.88
ITA	8.6	tweets test	**0.89**	0.68	0.77	0.71
SIA	8.6	imdb test	0.83	0.78	0.91	**0.84**
STA	8.6	tweets test	0.88	0.66	0.83	**0.72**

Table 8.3: Comparison of test results of fine-tuned and adapted models. Params = number of trainable parameters for evaluated dataset in thousands. Boldface indicates the best accuracy and F1 score for each dataset. ISF = IMDb base model with Skytrax fine-tuning, ITF = IMDb base model with Tweets fine-tuning, SIF = Skytrax base model with IMDb fine-tuning and STF = Skytrax base model with Tweets fine-tuning. ISA = IMDb base with Skytrax adapters, ITA = IMDb base with Tweets adapters, SIA = Skytrax base with IMDb adapters and STA = Skytrax base with Tweets adapters.

Overall the performances between the adapted and fine-tuned models are very similar across all the datasets. In table 8.3 can be seen which models performed the best on which datasets with regard to the number of trainable parameters used to fit the dataset in question.

From the number of trainable parameters a conclusion can be made that models using adapters are almost as good as fine-tuned models by a very small margin while having a much smaller number of trainable parameters, making them that much cheaper and faster to train.

Having concluded in section 8.1 that the fine-tuned models are equivalent in performance to the base models, it should be noted that with adapters we are getting two performant models for two datasets with almost no expensive calculation necessary while getting the two base models facilitates the need to train both from the ground up.

Model	**Params**	**Train time**	**Dataset**	**Accuracy**
BERT	110000	-	imdb test	**0.93**
DistilBERT	66000	90 hours	imdb test	**0.92**
SIF	1400	10 min	imdb test	0.84
SIA	8.6	6 min	imdb test	0.83

Table 8.4: Comparison of test results of proposed models to a third party benchmark. Params = number of trainable parameters for evaluated dataset in thousands. Boldface indicates the best accuracy. SIF = Skytrax base model with IMDb fine-tuning and SIA = Skytrax base with IMDb adapters.

8.4 Proposed Models vs Benchmark

As can clearly be seen from table 8.4 the fine-tuned and adapted models for the IMDb dataset are not as performant as the BERT and DistilBERT benchmarks from [11]. However it should be noted that the proposed SIA and SIF models have many orders of magnitude fewer parameters (tens of millions vs. thousands) and they take much less time to train (days vs minutes) while not being that much worse in accuracy (within 10 percentage points). This makes the proposed models a potentially viable alternative for scenarios where computational power is limited and having the absolute best accuracy in not necessary.

Conclusion

In conclusion, in this thesis the methods for performing sentiment analysis using transfer learning have been researched and explored. An approach using domain specific adapters inspired by Bapna et. al from 2019 [10] has been selected and a model architecture using a modified transformer has been proposed. This proposed model has been compared in multiple scenarios, firstly against a base variant of the model without the use of transfer learning, secondly against a fine-tuning variant of the same model and lastly against a third party benchmark using a state-of-the-art approach.

Experiments on all these models have been conducted using three datasets: movie reviews from IMDb, airline reviews from Skytrax and Twitter. The airline reviews have been selected for testing the in-domain knowledge transfer. These datasets have been used to train multiple combinations of base, fine-tuned and adapted models and evaluated on testing data.

From the obtained performance results, it has been concluded that the proposed models using adapters have similarly good performance compared to base models and fine-tuned models while having the advantage in being less computationally intensive. In comparison to a third party benchmark model by Sahn et. al from 2020 [11] the proposed models fell short in accuracy. However, by comparing the number of parameters of these proposed models it has been concluded that adapters are a viable alternative in low computational environments and they should be further studied and improved.

To conclude, the potential fields of further improvement are a usage of a bigger transformer architecture while training the base model on a different task. For example an unsupervised task of next word prediction could help the base model become more general and therefore improve the subsequent adapters.

Bibliography

[1] Medhat, W.; Hassan, A.; Korashy, H.: Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, ročník 5, č. 4, 2014: s. 1093 – 1113, ISSN 2090-4479, doi:https://doi.org/10.1016/j.asej.2014.04.011. Dostupné z: `http://www.sciencedirect.com/science/article/pii/S2090447914000550`

[2] Mikolov, T.; Chen, K.; Corrado, G.; aj.: Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013, s. 1–10. Dostupné z: `http://arxiv.org/abs/1301.3781`

[3] IBM: Neural network. Dostupné z: `https://1.cms.s81c.com/sites/default/files/2021-01-06/ICLH_Diagram_Batch_01_03-DeepNeuralNetwork-WHITEBG.png`

[4] Sigmoid plot. Dostupné z: `https://upload.wikimedia.org/wikipedia/commons/thumb/8/88/Logistic-curve.svg/2880px-Logistic-curve.svg.png`

[5] ReLU and GELU plot. Dostupné z: `https://upload.wikimedia.org/wikipedia/commons/thumb/4/42/ReLU_and_GELU.svg/2560px-ReLU_and_GELU.svg.png`

[6] Sarnovsky, M.: Convolutional neural network for text classification. Dostupné z: `https://www.researchgate.net/profile/Martin-Sarnovsky/publication/339851022/figure/fig2/AS:867996377559041@1583957871629/Convolutional-neural-network-for-text-classification.jpg`

[7] Vaswani, A.; Shazeer, N.; Parmar, N.; aj.: Attention Is All You Need. 2017, `1706.03762`.

[8] Devlin, J.; Chang, M.-W.; Lee, K.; aj.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Martinez, D.: Traditional machine learning vs Transfer learning. Dostupné z: `https://datascience.aero/wp-content/uploads/2020/03/transferlearning-119.jpg`

[10] Bapna, A.; Arivazhagan, N.; Firat, O.: Simple, Scalable Adaptation for Neural Machine Translation. 2019, `1909.08478`.

[11] Sanh, V.; Debut, L.; Chaumond, J.; aj.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. 2020, `1910.01108`.

[12] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556*, 09 2014.

[13] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. *CoRR*, ročník abs/1512.03385, 2015, `1512.03385`. Dostupné z: `http://arxiv.org/abs/1512.03385`

[14] Radford, A.; Wu, J.; Child, R.; aj.: Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 2018. Dostupné z: `https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf`

[15] Brown, T. B.; Mann, B.; Ryder, N.; aj.: Language Models are Few-Shot Learners. *CoRR*, ročník abs/2005.14165, 2020, `2005.14165`. Dostupné z: `https://arxiv.org/abs/2005.14165`

[16] Feldman, R.: Techniques and Applications for Sentiment Analysis. *Commun. ACM*, ročník 56, č. 4, apr 2013: str. 82–89, ISSN 0001-0782, doi:10.1145/2436256.2436274. Dostupné z: `https://doi.org/10.1145/2436256.2436274`

[17] Liu, B.: *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press, 2015, ISBN 1107017890,9781107017894.

[18] Bruce, R.; Wiebe, J.: Recognizing Subjectivity: A Case Study of Manual Tagging. *Natural Language Engineering*, ročník 5, 10 2000, doi:10.1017/S1351324999002181.

[19] Årup Nielsen, F.: A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. 2011, `1103.2903`.

[20] Hutto, C.; Gilbert, E.: VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. *Proceedings of the International AAAI Conference on Web and Social Media*, ročník 8, č. 1, May 2014: s. 216–225. Dostupné z: `https://ojs.aaai.org/index.php/ICWSM/article/view/14550`

[21] Koza, J. R.; Bennett, F. H.; Andre, D.; aj.: *Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming*. Dordrecht: Springer Netherlands, 1996, ISBN 978-94-009-0279-4, 151–170 s., doi:10.1007/978-94-009-0279-4_9. Dostupné z: `https://doi.org/10.1007/978-94-009-0279-4_9`

[22] Russell, S.: *Artificial intelligence : a modern approach*. Hoboken, NJ: Pearson, 2021, ISBN 0-13-461099-7.

[23] Pang, B.; Lee, L.; Vaithyanathan, S.: Thumbs up? Sentiment Classification Using Machine Learning Techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, USA: Association for Computational Linguistics, 2002, str. 79–86, doi:10.3115/1118693.1118704. Dostupné z: `https://doi.org/10.3115/1118693.1118704`

[24] LIU, L.; ÖZSU, M. T. (editoři): *Encyclopedia of Database Systems*. Springer US, 2009, doi:10.1007/978-0-387-39940-9. Dostupné z: `https://doi.org/10.1007/978-0-387-39940-9`

[25] Haddi, E.; Liu, X.; Shi, Y.: The Role of Text Pre-processing in Sentiment Analysis. *Procedia Computer Science*, ročník 17, 2013: s. 26–32, ISSN 1877-0509, doi:https://doi.org/10.1016/j.procs.2013.05.005, first International Conference on Information Technology and Quantitative Management. Dostupné z: `https://www.sciencedirect.com/science/article/pii/S1877050913001385`

[26] Green, T. J.: *Bag Semantics*. Springer US, 2009, 201–206 s., doi:10.1007/978-0-387-39940-9_979. Dostupné z: `https://doi.org/10.1007/978-0-387-39940-9_979`

[27] El-Khair, I. A.: *TF - IDF*. Springer US, 2009, 3085–3086 s., doi:10.1007/978-0-387-39940-9_956. Dostupné z: `https://doi.org/10.1007/978-0-387-39940-9_956`

[28] Ng, A.: NIPS 2016 tutorial: "Nuts and bolts of building AI applications using Deep Learning" by Andrew Ng. 2016.

[29] Pan, S. J.; Yang, Q.: A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, ročník 22, č. 10, 2010: s. 1345–1359, doi:10.1109/TKDE.2009.191.

[30] Liu, R.; Shi, Y.; Ji, C.; aj.: A Survey of Sentiment Analysis Based on Transfer Learning. *IEEE Access*, ročník 7, 2019: s. 85401–85412, doi: 10.1109/ACCESS.2019.2925059.

[31] Dai, W.; Yang, Q.; Xue, G.-R.; aj.: Boosting for Transfer Learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, New York, NY, USA: Association for Computing Machinery, 2007, ISBN 9781595937933, str. 193–200, doi:10.1145/1273496.1273521. Dostupné z: `https://doi.org/10.1145/1273496.1273521`

[32] Donges, N.: What Is Transfer Learning? Exploring the Popular Deep Learning Approach. Dostupné z: `https://builtin.com/data-science/transfer-learning`

[33] Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; aj.: Parameter-Efficient Transfer Learning for NLP. 2019, `1902.00751`.

[34] Tao, J.; Fang, X.: Toward multi-label sentiment analysis: a transfer learning based approach. *Journal of Big Data*, ročník 7, č. 1, Leden 2020, doi:10.1186/s40537-019-0278-0. Dostupné z: `https://doi.org/10.1186/s40537-019-0278-0`

[35] Smetanin, S.; Komarov, M.: Deep transfer learning baselines for sentiment analysis in Russian. *Information Processing & Management*, ročník 58, č. 3, 2021: str. 102484, ISSN 0306-4573, doi:https://doi.org/10.1016/j.ipm.2020.102484. Dostupné z: `https://www.sciencedirect.com/science/article/pii/S0306457320309730`

[36] Maas, A. L.; Daly, R. E.; Pham, P. T.; aj.: Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA: Association for Computational Linguistics, June 2011, s. 142–150. Dostupné z: `http://www.aclweb.org/anthology/P11-1015`

[37] Tripathi, S.; Mehrotra, R.; Bansal, V.; aj.: Analyzing Sentiment using IMDb Dataset. In *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2020, s. 30–33, doi: 10.1109/CICN49253.2020.9242570.

[38] Shaukat, Z.; Zulfiqar, A. A.; Xiao, C.; aj.: Sentiment analysis on IMDB using lexicon and neural networks. *SN Applied Sciences*, ročník 2, č. 2, Leden 2020, doi:10.1007/s42452-019-1926-x. Dostupné z: `https://doi.org/10.1007/s42452-019-1926-x`

[39] Ali, N. M.; Abd El Hamid, M. M.; Youssif, A.: Sentiment Analysis for Movies Reviews Dataset Using Deep Learning Models. *International*

Journal of Data Mining & Knowledge Management Process, ročník 2, č. 2, Červen 2019. Dostupné z: `https://ssrn.com/abstract=3403985`

[40] Skytrax: Quality is our journey. Dostupné z: `https://skytraxresearch.com/`

[41] Efehan: Skytrax Airline Reviews. Dostupné z: `https://www.kaggle.com/efehandanisman/skytrax-airline-reviews`

[42] Liu, H.; Yin, Q.; Wang, W. Y.: Towards Explainable NLP: A Generative Explanation Framework for Text Classification. *CoRR*, ročník abs/1811.00196, 2018, `1811.00196`. Dostupné z: `http://arxiv.org/abs/1811.00196`

[43] Eight, F.: Twitter us airline sentiment. Oct 2019. Dostupné z: `https://www.kaggle.com/crowdflower/twitter-airline-sentiment`

[44] Rustam, F.; Ashraf, I.; Mehmood, A.; aj.: Tweets Classification on the Base of Sentiments for US Airline Companies. *Entropy*, ročník 21, č. 11, 2019, ISSN 1099-4300, doi:10.3390/e21111078. Dostupné z: `https://www.mdpi.com/1099-4300/21/11/1078`

[45] Wan, Y.; Gao, Q.: An Ensemble Sentiment Classification System of Twitter Data for Airline Services Analysis. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015, s. 1318–1325, doi:10.1109/ICDMW.2015.7.

[46] Kumar, S.; Zymbler, M.: A machine learning approach to analyze customer satisfaction from airline tweets. *Journal of Big Data*, ročník 6, č. 1, Červenec 2019, doi:10.1186/s40537-019-0224-1. Dostupné z: `https://doi.org/10.1186/s40537-019-0224-1`

[47] Van Rossum, G.; Drake, F. L.: *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN 1441412697.

[48] Harris, C. R.; Millman, K. J.; van der Walt, S. J.; aj.: Array programming with NumPy. *Nature*, ročník 585, č. 7825, Září 2020: s. 357–362, doi:10.1038/s41586-020-2649-2. Dostupné z: `https://doi.org/10.1038/s41586-020-2649-2`

[49] pandas development team, T.: pandas-dev/pandas: Pandas. Únor 2020, doi:10.5281/zenodo.3509134. Dostupné z: `https://doi.org/10.5281/zenodo.3509134`

[50] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; aj.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, ročník 12, 2011: s. 2825–2830.

[51] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org. Dostupné z: `https://www.tensorflow.org/`

[52] Chollet, F.; aj.: Keras. 2015. Dostupné z: `https://github.com/fchollet/keras`

[53] Bisong, E.: *Google Colaboratory*. Berkeley, CA: Apress, 2019, ISBN 978-1-4842-4470-8, s. 59–64, doi:10.1007/978-1-4842-4470-8_7. Dostupné z: `https://doi.org/10.1007/978-1-4842-4470-8_7`

[54] Keras: Keras documentation: Text classification with Transformer. Dostupné z: `https://keras.io/examples/nlp/text_classification_with_transformer/`

[55] Keras: Keras documentation: Embedding layer. Dostupné z: `https://keras.io/api/layers/core_layers/embedding/`

Appendix A

List of Abbreviations

API	Application Programming Interface
BOW	Bag of Words
CNN(s)	Convolutional Neural Network(s)
CPU	Central Processing Unit
FFN(s)	Feed-forward Network(s)
GELU	Gaussian Error Linear Unit
GPU	Graphics Processing Unit
LSTM	Long Term Short Memory
ML	Machine Learning
NLP	Natural Language Processing
NN(s)	Neural Network(s)
ReLU	Rectified Linear Unit
RNN(s)	Reccurent Neural Network(s)
TL	Transfer Learning
TPU	Tensor Processing Unit

Appendix **B**

Contents of Enclosed CD

```
readme.md ........................ the file with CD contents description
├── experiments ....... the directory with the notebooks for the experiments
├── preprocessing the directory with the notebooks for the preprocessing of
│   datasets
├── statistics ..... the directory with the notebooks for datasets statistics
├── thesis ................. the directory of LaTeX source codes of the thesis
└── thesis.pdf ...................................... the thesis in PDF
```