Stian Steinbakken

Paying Attention to Native-Language Identification

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Stian Steinbakken

Paying Attention to Native-Language Identification

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Abstract

Native-language identification (NLI) is the task of identifying an author's first language (L1) given only information expressed in the author's second language (L2). The task of NLI is important for educational purposes, as learning what errors a student typically makes when writing or speaking in a foreign language can help educators identify common patterns, and produce custom learning experiences based on different L1s. NLI also has applications in forensic science, author profiling, web-scraping, data collection, as well as detecting web fraud and grooming.

Previous solutions to solve the task of NLI have mostly relied on traditional machine learning techniques, with Support Vector Machines being the most popular choice. These traditional classifiers have been shown to consistently outperform deep learning based approaches on the task – possibly due to the lack of labeled data. However, attention-based deep learning systems have recently provided state-of-the-art results for a large number of natural language processing tasks, and have become ubiquitous in sequence processing. Attention-based systems do not rely on any form of recursion or convolution, which allows for heavy parallelization and fast training of large models.

With the recent success of attention-based systems, as well as more labeled NLI data now available, this Master's Thesis explores how such attention-based systems can be used to increase performance on the task of NLI. In particular, an experimental research approach is applied to empirically explore how the attention-based system BERT – Bidirectional Encoder Representations from Transformers – can be used to obtain state-of-the-art results in NLI.

BERT is first tested in isolation on the TOEFL11 data set, which has been the de facto standard data set for training and testing of NLI systems since 2013. Next, the model is tested on the novel, much larger Reddit-L2 data set, and is shown to produce state-of-the-art results. BERT is then used in a meta-classifier stack in combination with traditional techniques to produce an accuracy of 0.853 on the TOEFL11 test set. Finally, BERT is trained on more than 50 times as many examples as have been used for English NLI before, to produce an accuracy of 0.902 on the Reddit-L2 in-domain test scenario – beating the previous state-of-the-art by 21.2 percentage points.

Sammendrag

Morsmålsidentifisering går ut på å identifisere en forfatters eller talers morsmål (L1) basert på tekster eller opptak gjort av denne personen på et annet språk (L2). Morsmålsidentifisering kan være viktig for pedagogikk, da å vite hvilke feil en elev med et gitt morsmål typisk gjør når han eller hun skal lære et annet språk kan hjelpe pedagoger med å kjenne igjen vanlige mønster og lage tilpassede læringsopplegg basert på forskjellige språk. Morsmålsidentifisering har også bruksområder innenfor kriminalteknikk, forfattergjenkjenning og datainnsamling, i tillegg til å kunne brukes til å detektere digitalt bedrageri og barnelokking.

De tidligere beste løsningene for morsmålsidentifisering har i hovedsak basert seg på tradisjonelle maskinlæringsteknikker, primært støttevektormaskiner. Disse tradisjonelle teknikkene har gang på gang gitt bedre resultater enn løsninger som baserer seg på dyp-læring – trolig på grunn av mangel på annotert data for morsmålsidentifisering. Oppmerksomhetsbaserte dyp-læring teknikker har dog nylig blitt allestedsnærværende i sekvensprosessering, og oppnår tidenes beste resultater på flere oppgaver innenfor naturlig språkbehandling. Disse oppmerksomhetsbaserte systemene krever ingen rekursjon eller konvolusjon, som gjør at systemene kan parallelliseres i stor grad og gjør det mulig å trene store modeller raskt.

Med bakgrunn i den imponerende ytelsen til oppmerksomhetsbaserte dyplæring teknikker, i tillegg til at det nå finnes mer annotert data, utforsker denne masteroppgaven hvordan slike oppmerksomhetsbaserte dyp-læring teknikker kan anvendes for å øke ytelsen i morsmålsidentifisering. Mer spesifikt ser denne oppgaven nærmere på hvordan det oppmerksomhetsbaserte systemet BERT – Bidirectional Encoder Representations from Transformers – kan brukes, både alene og i kombinasjon med eksisterende teknikker, for å oppnå bedre resultater innen morsmålsidentifisering.

BERT taes først i bruk på TOEFL11 datasettet, som har vært standard datasett for morsmålsidentifisering siden 2013. Deretter anvendes BERT på det langt større datasettet kalt Reddit-L2, hvor modellen oppnår "state-of-the-art" resultater. Videre brukes BERT, sammen med tradisjonelle teknikker under en meta-klassifikator, til å oppnå en treffsikkerhet på 0.853 på TOEFL11 testsettet. BERT trenes så på mer enn 50 ganger så mye data som har blitt brukt for engelsk morsmålsidentifisering tidligere, og produserer en treffsikkerhet på 0.902 på det såkalte *Reddit-L2 in-domain* scenarioet – 21.2 prosentpoeng bedre enn den tidligere beste treffsikkerheten oppnådd.

Preface

This Thesis is submitted as the final part of the work to achieve the degree of Master of Science in Computer Science from the Norwegian University of Science and Technology (NTNU). The work was done at the Department of Computer Science and was supervised by Björn Gambäck.

A special thanks goes out to Björn for suggesting the Thesis subject, and for helping making the correct decision throughout the process of writing the Thesis. A thanks also goes out to Hans Olav Slotte for his previous work on his Master's Thesis on the subject of NLI, as well as Vebjørn Isaksen for good discussions on the topic of BERT and implementation details. Furthermore, acknowledgements go out to Rabinovich et al. [2018] for making the Reddit-L2 data set openly available online, and to Devlin et al. [2018] for distributing the pre-trained BERT models and making the code openly available. Such openness helps Artificial Intelligence progress as a field much faster than what would be possible without. An additional thanks goes to Shervin Malmasi and Mark Dras of Malmasi and Dras [2018] for answering questions about the details of their LDA classifier. Furthermore, a thanks goes out to the HPC group at NTNU, for allowing the use of the Idun cluster. The experiments of this Thesis would never have been possible without these resources. Finally, a special thanks goes out to to Jacob Devlin, Ian Goodfellow, Jay Alammar, Aidan Gomez and Jakob Uszkoreit for permitting the use of figures from their books and papers.

Stian Steinbakken
Trondheim, June 11, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Native-language identification (NLI) is the task of identifying an author's first language (L1) given only information expressed in the author's second language (L2) [Malmasi and Dras, 2018]. For instance, given an *English* text or voice recording by a person natively from *Spain*, the task is to identify that the author is natively *Spanish*. In recent years native-language identification has blossomed as a field within natural language processing (NLP).

NLI operates under the assumption that an author's primary language will dispose them towards particular language production patterns in their secondary language [MacDonald, 2013]. This relates to cross-linguistic influence (CLI), a key topic in the field of second-language acquisition (SLA) that analyzes transfer effects from the L1 on later learned languages. The task of NLI is usually treated as a supervised classification problem, requiring labeled data. Until recently the TOEFL11 [Blanchard et al., 2013] data set has been the standard corpus for NLI, concerning English learners of 11 different L1s. However, recent advances have been made in auto-generating data based on social media users of high proficiency [Goldin et al., 2018], resulting in a much larger data set with more languages to classify. Current state-of-the-art approaches on the TOEFL11 data set reach over 0.88 accuracy in identifying the native language of the author using text only, while the best results on more English-proficient social media users approaches 0.69 when evaluated on the same topics as trained on [Goldin et al., 2018].

While the field of NLI is evolving, so are advances in deep learning. Recently, attention-based deep learning models have shown promising results on various NLP tasks, and has become the de facto state-of-the-art in sequence-to-sequence processing. Novel attention models, such as the Transformer [Vaswani et al., 2017], rely solely on attention-mechanisms. Attention allows for heavy parallelization in the training of the system, as no recurrence or convolution is

required, while still yielding state-of-the-art results.

BERT – Bidirectional Encoder Representations from Transformers – is a novel language representation model based on Transformers. Using a network of transformers, BERT obtains new state-of-the-art results on 11 natural language processing tasks [Devlin et al., 2018].

Given the promising results of attention-based systems and the need for good NLI systems, the main focus of this Master's Thesis will be to explore how these attention-based systems can be used to improve performance on the task of NLI.

1.1 Background and Motivation

NLI has applications both within second language acquisition and education, as well as forensic linguistics and web-scraping.

Learning what mistakes an L1 speaker typically makes when writing or speaking in a foreign language, L2, can help educators create custom tutoring experiences and give feedback based on L1s, which can potentially help students learn better and faster.

Forensics linguistics is the application of linguistics and science to aid in criminal investigation. The field is a sub-branch of author identification, and NLI can be a helpful tool in identifying what country an author is originally from. An example is identifying texts of unknown authors. If culprits have left a ransom note at a crime scene, NLI can help identify what country the perpetrators are from, reducing the required search space and further aid the investigation. NLI also has uses in other fields related to author profiling, such as plagiarism control, which could be further extended to detecting web fraud and grooming.

Additionally, Goldin et al. [2018] argue that most of the English textual content on the Internet – the source of many of the machine learning corpora used today – is created by non-native English speakers. This makes the task of identifying whether an author is native or not relevant. With this in mind, NLI can be useful in web-applications, web-crawling and data collection to ensure high quality data.

As mentioned in the introduction, attention-based learning models have shown very promising results in NLP recently. BERT obtains state-of-the-art results in tasks varying from question answering to sentence classification. These results, along with the possibility for heavy parallelization and optimization, make attention-based systems an interesting candidate for solving the task of NLI.

1.2 Goals and Research Questions

This Master's Thesis has one main goal, which will be described below. In order to reach this goal, three Research Questions have been composed. Answering the three Research Questions should help concretize and reach the overall main goal.

Goal *Explore how attention-based architectures can be used to improve performance on the task of Native-Language Identification.*

Given recent advancements in performance of attention-based systems on various natural language processing (NLP) tasks, the main goal of this Master's Thesis will be to explore how these rather novel architectures can be used to improve performance on the task of NLI. More specifically, the goal will be to explore how BERT – a Transformer-based architecture built on attention – can be used for the task.

Research Question 1 *How do attention-based systems perform compared to the current state-of-the-art with regards to the task of NLI?*

The first Research Question will dwell on how the attention-based system alone compares to the current state-of-the-art. Answering Research Question 1 will help isolate and evaluate an attention-based system alone, before exploring how such a system can be used to improve existing approaches.

Research Question 2 *How robust are attention-based systems when tested on different topics than that which the systems are trained on?*

Today's state-of-the-art systems are known for quite substantial drops in performance when being tested on documents about topics which they have not seen during training [Malmasi and Dras, 2018]. Research Question 2 will aim to explore the robustness of an attention-based system across different topics, and compare such a system to the current state-of-the-art.

Research Question 3 *Can attention-based systems, in combination with techniques used in the current state-of-the-art, improve performance on the task of NLI?*

While the attention-based system might perform well on its own, it will also be of interest to explore how the system allows for improvements in combination with existing techniques. All of the best NLI systems to date include some sort of multi-classifier or ensemble based architecture. Answering Research Question 3 will enable the exploration of how attention-based systems can be used in such ensembles to improve performance on the task of NLI.

1.3 Research Method

In order to achieve the goal of the Master's Thesis, an experimental research approach will be applied. Experiments using the same test data as related literature will be carried out, evaluating both attention-based systems, current state-of-the-art techniques, and the combination of the two.

1.4 Contributions

The main contributions of this Master's Thesis are as follows:

1. *A collection of experiments using BERT on the TOEFL11 data set, which show that BERT alone is not able to compete with the traditional state-of-the-art approaches for NLI on the TOEFL11 test set.*

2. *A meta-classifier architecture which uses both BERT and traditional classifiers as base classifiers, which produces an accuracy of **0.853** on the TOEFL11 test set – closing in on the current state-of-the-art of 0.882.*

3. *A simple, yet effective scheme for dividing large documents for BERT, based on heuristically splitting training documents into sub-documents, and using a majority vote to recombine the predictions of the model.*

4. *State-of-the-art results using BERT on the novel Reddit-L2 data set, both in- and out-of-domain, producing a **0.902** accuracy on the 10-fold cross-validation in the Reddit in-domain scenario – a 21.2 percentage point improvement over the previous state-of-the-art.*

5. *An exploration of BERT's robustness when trained on different topics than tested on. This in form of experiments using BERT alone, as well as in a meta-classifier stack, the latter which produces a final accuracy of **0.529** in the Reddit-L2 out-of-domain test scenario – a 16.7 point improvement over the previous out-of-domain state-of-the-art.*

6. *An exploration of ensembles and meta-classifiers for NLI, which shows that the inclusion of BERT in these classifier stacks can improve the final accuracy with up to 10 percentage points, depending on the task and base-classifiers used.*

7. *A large experiment using 1.4 million, out-of-domain training documents, for which BERT obtains a final F_1 score of **0.847** on the entire Reddit-L2 in-domain data – a test set which contains more than 140 times as many test cases as the original in-domain scenario.*

1.5 Thesis Structure

The Master's Thesis is structured in the following manner:

1. Chapter 2 provides all the background theory necessary to understand the rest of the Thesis. The chapter includes topics such as general machine learning, machine learning models, deep learning, attention and natural language processing.

2. Chapter 3 covers the related work carried out in the field of NLI and the BERT model architecture. In addition, a motivation section is provided in order to relate the findings of the literature review to the experiments of the Thesis.

3. Chapter 4 covers the model architectures which will be used for the different experiments.

4. Chapter 5 provides an experimental plan and the experimental setup used for all experiments. The chapter ends with the results of all experiments which have been performed.

5. Finally, chapter 6 evaluates and concludes the merits of the Master's Thesis in light of the results and findings of the experiments, and provides suggestions for possible future work.

Chapter 2

Background Theory

In order to understand the field of NLI and attention, some background knowledge in machine learning, natural language processing and deep learning is required. This chapter will cover all of the concepts needed in order to understand the content discussed in the remaining chapters. The background theory section has been created in the following manner: When writing the Thesis, every time a term or concept appears which is deemed to be within the scope of the Thesis – and not assumed to be known beforehand by the reader – a section is created in this chapter which covers said concept or term. This method is applied to ensure that all background knowledge required to get a full understanding of the Thesis has been covered, and to ensure that the background section does not contain unnecessary information not relevant with regards to the rest of the Thesis. The background theory section assumes that the reader is already familiar with mathematical fields such as linear algebra and statistics.

2.1 Machine Learning

Machine Learning is a large field within Artificial Intelligence and is the study and exploration of algorithms that can "learn" and make predictions on data, without being explicitly programmed to do so [Goodfellow et al., 2016]. These machine learning algorithms are typically divided into supervised and unsupervised learning, though there exist other approaches. This Thesis will primarily focus on the supervised learning algorithms, as the vast majority of work that has been done on NLI treats it as a supervised learning problem. The goal of machine learning is to use these learning algorithms to create a model which can generalize and predict future, unseen data samples.

Supervised learning algorithms require labeled data, meaning that each ex-

ample is labeled with the correct answer. If the goal is to predict whether an image is an image of a cat or a dog, supervised learning algorithms require that each image in the data set has a label marking it as either of the two classes. The model will learn from this data, but it must also be assessed and evaluated in a correct manner. For this reason, the data is typically split into three parts: a training set, a development set (often also called validation set), and a test set. The **training set** is typically the largest of the three, and contains the data the model is allowed to train on. If the model was a student, the training set would be example tasks handed out by the professor before an exam, with known answers and solutions. The student would use these examples to learn how to solve the example tasks when knowing the correct answer. Typically the training set constitutes between 80% and 90% of the available data. The final partition, the **test** set, can be viewed as the exam itself. Just as in real life, the student does not train and learn during the exam, and the model will never see the test set before the final evaluation. The development or validation set is mainly used for *hyper-parameter tuning* and meta-optimization, which means tuning the parameters the model takes as input. The validation set can loosely be viewed as taking a practice exam before taking the actual exam which the student will be tested on, in order to decide what parameters will help the model perform optimally on the final test set.

2.1.1 Classification Tasks

In machine learning, a classification task is the problem of deciding to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known. In most of the existing literature, NLI is treated as a classification task. In the case of NLI, the observation is a text or document written in an authors L2, and the category (or class) it belongs to is the L1 of the author.

2.1.2 Features and Feature Space

Prudent concepts in machine learning are *features* and *feature space*. Features are simply the attributes of the data. Given a data set which consist of cats and dogs, features could be the length of the nose, or the weight of the animal. Depending on the data, perhaps dogs weigh more than cats, and dogs have longer noses. Given the correct labels the model will be able to learn such relations, and classify future examples correctly. Features are typically represented as vectors. If the first axis is defined to be the length of the nose of the training sample, and the second axis to be the weight, then an animal that has a 10.2 cm long nose and weighs 23 kg would be represented as a vector equal to $\begin{bmatrix} 10.2 \\ 23 \end{bmatrix}$. This is a

simple example with only two dimensions, but the same principle applies as the number of attributes and dimensions increase.

The feature space is the vector room of the feature vectors. The higher dimension of the feature vectors, the larger the feature space, and thus the more possible feature-instances there are.

2.1.3 Feature Representations

In the above cats and dogs example, the raw attributes were fed into the feature vector representing the animal. However, some manipulation of the feature vectors can map the vectors into a representation which is more meaningful to a machine learning model. Some relevant techniques for achieving this will be covered below.

Normalization

Normalization in the context of feature vectors means mapping real values into values which are meaningful to the model. Typically this consists of mapping unbound values to values between 0 and 1, or having all the values sum up to 1 in total. Among many others, a popular normalization technique in machine learning is L2 normalization, which ensures that the *squared* values of all the values in a vector sum to one.

One-hot-encoding

In classification tasks, a model is trained to recognize which category (or class) a sample belongs to. A useful way of encoding the class to represent is via one-hot-encoding. Given a set of classes – "cat", "dog" and "fish" – a way of encoding this mathematically could be to say that a label of 0 corresponds to cat, 1 corresponds to dog, and 2 corresponds to fish. Mathematically this would imply that dog has a larger value than cat, and that a fish has a larger value than both cat and dog. This sort of value ordering makes little sense, and a better way of representing a class is via one-hot-encoding. In a one-hot vector, all values are set to zero except the value at the index of the represented class. For example, with the same ordering as described above, the vector for cat, dog and fish would respectively be represented as

- Cat = $\begin{bmatrix}1 & 0 & 0\end{bmatrix}$
- Dog = $\begin{bmatrix}0 & 1 & 0\end{bmatrix}$
- Fish = $\begin{bmatrix}0 & 0 & 1\end{bmatrix}$

One-hot-encoding alleviates the model of the burden of learning an artificial ordering of the class labels, which typically improves performance of the model. It is a common technique for representing classes and scales to an infinite number of classes.

TF-IDF Weighting of Features

TF-IDF is a concept from the field of Information Retrieval, and stands for term frequency-inverse document frequency [Spärck Jones, 1972]. TF-IDF is a numerical statistic that is intended to reflect how important a word, or term, is to a document in a corpus [Huq et al., 2018]. TF-IDF consists of the product of two numerical terms, namely the term frequency (TF) and the inverse document frequency (IDF). Given a term t and a document d, the term frequency is simply how frequently the term appears in the document. This can be expressed in multiple ways, for instance binary ($\text{tf}(t, d) = 1$ if the term appears in the document, 0 otherwise), or the percentage of times the term appears. I. e., $\text{tf}(t, d) = f_{t,d}/T$ where $f_{t,d}$ is the number of times the term appears in the document, and T is the total number of terms in the document.

The IDF is a measure of how much information the term provides. For example function words like "the" and "a" will have a high TF as they appear a lot in most documents, but they provide little discriminating power. Common terms like these are "promiscuous", as they usually appear frequently in all documents. IDF is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

$$\text{IDF}(t, D) = \log(\frac{N}{1 + \text{num}_{t,d \in D}}) \tag{2.1}$$

where N is the total number of documents, $\text{num}_{t,d \in D}$ is the total number of documents where the term t appears, and the $+1$ is a smoothing constant for avoiding division by zero if a term does not appear in any documents. IDF makes intuitive sense as a term that appears in a lot of documents will have a low IDF, but terms that only appear in a few selected documents (and thus is assumed to have a higher discriminative power) will have a high value. Combining the two concepts yields

$$\text{TF-IDF}(t, d, D) = \text{tf(t, d)} \cdot \text{IDF}(t, D) \tag{2.2}$$

which will have its highest value when t occurs many times within a small number of documents, lower when the term occurs fewer times in a document or occurs in many documents, and its lowest value when the term occurs frequently in all documents.

With regards to features, TF-IDF can be used to *weigh* each term in a vocabulary, which means weighting each entry with its corresponding TF-IDF score. For example, if each entry in a feature vector corresponds to one word in the vocabulary, one way of representing a document is to simply count how many times each word appears in the document, and set the corresponding value in the feature vector to be this count. Depending on the vocabulary and document, this vector would be rather sparse and might not carry much information for the model to use. A more useful representation of this feature vector would be to weigh each entry in the vector by the TF-IDF score of each word. This encapsulates global information in each feature entry, and is a common feature weighting and normalization scheme in many machine learning applications.

2.1.4 Ensemble Learning and Meta-Classifiers

Ensemble learning is the combination of several different models into one, hopefully better, model [Opitz and Maclin, 1999]. Intuitively, this can be viewed as each different model having a set of hypotheses about the data. By having multiple models, not only can the total number of possible hypotheses be increased, but the combination of these hypotheses can yield new hypotheses which were not present pre assembling.

There exist many different ways of performing ensemble learning. One of the most straight forward methods for ensemble learning is called bootstrap aggregating, or *bagging*, where each model is trained on a unique subset of the full training set.

Furthermore, once an ensemble has been built, there are several methods for combining the outputs of the models into a single output. Examples include *majority voting* where the label predicted by most of the models is selected as the unified decision, as well as the *mean probability decision*, where each model outputs the probability or certainty for each class, and the class with the highest average probability across all classifiers is selected.

Meta-classifiers, or classifier-stacking, is a similar technique to ensemble learning, and concerns training classifiers on the outputs of other classifiers. It is an advanced method that has proven to be effective in many classification tasks [Malmasi and Dras, 2018]. Ensemble architectures can be both homogeneous, where all classifiers in the ensemble have the same type, structure and parameters, as well as heterogeneous where the classifiers can be different from each other.

2.2 Models and Learning Machines

As the previous section covered the basics of machine learning, this section will describe some traditional statistical models and learning machines which are commonly used both in NLI and in machine learning in general.

2.2.1 Logistic Regression

Logistic Regression is a technique from statistics which can be used in machine learning for classification tasks. The maths behind logistic regression will not be covered here, only a basic intuition of what the method does.

When using *linear* regression, the dependent variable is continuous and can therefore be estimated using methods such as the least square method. In tasks like classification, however, the dependent variable is discrete and categorical, and methods such as least squares will fail. *Logistic* regression is a method for binary classification, which aims to fit an objective function to the binary data points in order to maximize the probability that the curve will correctly predict the data [Goodfellow et al., 2016]. This maximum likelihood estimation is done in an iterative manner. When using logistic regression for a classification task which is not binary (i.e., there are more than two categories), one can either use several logistic regression models and compare each output, or use an extension of logistic regression called *multinomial* logistic regression, which can handle multiple categories. A positive attribute of logistic models is that the classification is only a byproduct of predicting the probability of each class. This means that logistic regression models can not only yield the most probable classification, but also *how* probable or certain the model is that the given sample belongs to the predicted class.

2.2.2 Naive-Bayes

Naive-Bayes classifiers [McCallum and Kamal, 1998] is not a single type of classifier, but a family of simple probabilistic classifiers operating under the same probabilistic assumption. Naive-Bayes models assume that all features are independent from each other, and applies Bayes Theorem based on this assumption. Naive-Bayes models differentiate themselves from other popular models, as no iterative approximation is needed and the calculations required can be performed in linear time. This makes Naive-Bayes models highly scalable and fast.

Given a class label y and a feature vector $\vec{x}$, the probability of the class given the input vector under Bayes theorem can be written as

$$P(y|\vec{x}) = \frac{P(\vec{x}|y)P(y)}{P(\vec{x})} \tag{2.3}$$

The vector $\vec{x}$ can be written as $\vec{x} = (x_1, x_2 \dots x_n)$. By substituting $\vec{x}$ and expanding using the chain-rule the same equation can be written as

$$P(y|\vec{x}) = \frac{P(\vec{x_1}|y)P(x_2|y)\dots P(x_n|y)P(y)}{P(x_1)P(x_2)\dots P(x_n)} \tag{2.4}$$

As the denominator is constant for all input vectors, the equality can be replaced with proportionality and the equation can be written as:

$$P(y|\vec{x}) \propto P(y)\prod_{i=1}^{n} P(x_i|y) \tag{2.5}$$

In the multiclass case, the problem can be reduced to finding the most probable class. That is:

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y) \tag{2.6}$$

Finally, the probabilities $P(x_i|y)$ must be found. There are several ways to calculate these values, but for classification problems it is common to draw these values from a multinomial distribution. This is known as the multinomial Naive Bayes classifier. To this day, Naive-Bayes classifiers still remain a popular baseline method for text categorization [McCallum and Kamal, 1998].

2.2.3 Support Vector Machines

A Support Vector Machine (SVM) is a model for supervised learning which aims to separate positive and negative samples in a feature space using a hyperplane. The basic idea is to find the extremes of both the positive and negative samples, and then find the hyperplane separating these extreme samples with the widest margin. These boundaries are called the support vectors. Any arbitrary hyperplane that separates all samples would in theory do, but SVMs aim to mathematically optimize the distance of the hyperplane to the closest samples from both groups of samples, in order to maximize the probability that future, unseen samples end up on the correct side of the hyperplane. In order to maximize the distance from the hyperplane to the support vector, SVMs use a *kernel*. A kernel, in this context, is a mathematical function which defines the similarity between two vectors. In regular linear SVMs, the kernel function will simply be the inner product of the two sample vectors, which is the similarity between two vectors in the original space, represented as a scalar. The idea of Support Vector Machines emerged in Cortes and Vapnik [1995], though Cortes and Vapnik had allegedly been working on the idea since the 1960s. Today SVMs are used for a wide range of tasks in machine learning, and have proven very powerful in many classification tasks.

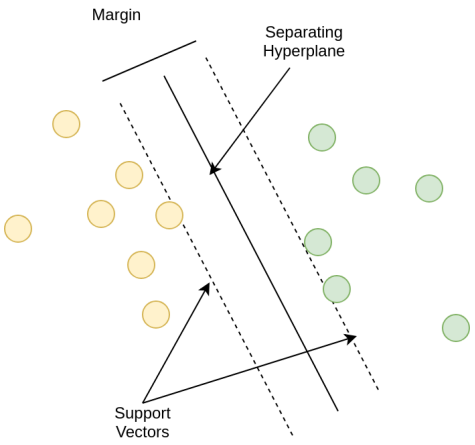

Figure 2.1: A visualization of a Linear Support Vector Machine in two dimensions, separating green and yellow data points.

A 2D visualization of a linear SVM can be seen in figure 2.1. Linear Support Vector Machines (who use the inner product as their kernel function) will be able to separate data which is linearly separable. However, linear SVMs will struggle when the data is **not** linearly separable. In order to combat this problem, a method called the kernel trick is applied. The kernel trick is to change the kernel function used – i.e., how the similarity between to samples is defined — in order to map the data samples to a higher dimensional space, where the samples actually are linearly separable. There exist several kernel functions, including the polynomial kernel which maps the feature space into a higher dimension polynomial, as well as others such as the Radial Basis Function (RBF) and the Gaussian kernel function. Similarly to logistic regression, one can obtain the certainty or probability that a given sample belongs to a certain class by looking at the distance of the sample to the separating hyperplane.

String Kernels

A string kernel is a kernel function which measures the similarity of two strings. Strings in this context are defined as a finite sequence of symbols that need not be of the same length. A higher value as output indicates a higher similarity. String kernels allow kernel based approaches such as SVMs to operate on strings, without having to map the input strings into real-valued vectors of a fixed length [Lodhi et al., 2002].

The kernel is the inner product in the feature space generated by all sub-sequences of length n. These sub-sequences consist of any ordered sequence of length n found in the text, and does not need to be contiguous [Lodhi et al., 2002]. Less formally explained, these sequences can be viewed as character n-grams (see section 2.5.2), but with the property that they do not need to appear next to each other in the text as regular n-grams do. The idea is that the more sub-strings two documents have in common, the more similar they are. In order to deal with the fact that these sub-strings do not need to be contiguous, Lodhi et al. [2002] introduce an exponentially decaying weighting factor, $\lambda \in (0, 1)$, who's exponent increases the further apart the characters are.

An example would be the sub-string "c-a-r", which is present both in the word "**car**d" and in the word "**c**ust**ar**d", but with different weighting. For each such substring there is a dimension of the feature space, and the value of the coordinate depends on how frequently and how compactly the string is embedded in the text.

As a concrete example from [Lodhi et al., 2002] with $n = 2$, consider four documents which contain one word each: "cat", "car", "bat", "bar". There are eight different possible sub-strings across all these documents (c-a, c-t, a-t, b-a, b-t, c-r, a-r, b-r), which means that the kernel function will operate in a 8-dimensional space. The words would be mapped as shown in table 2.1, where ϕ is the kernel function and the columns contain the value for each possible sub-string.

	c-a	c-t	a-t	b-a	b-t	c-r	a-r	b-r
ϕ(cat)	λ^2	λ^3	λ^2	0	0	0	0	0
ϕ(car)	λ^2	0	0	0	0	λ^3	λ^2	0
ϕ(bat)	0	0	λ^2	λ^2	λ^3	0	0	0
ϕ(bar)	0	0	0	λ^2	0	0	λ^2	λ^3

Table 2.1: An example of the kernel values for each document (row) with regards to each possible sub-string (columns) using string kernels.

Note that $\lambda^2 > \lambda^3$ since $\lambda \in (0, 1)$, so a lower exponent for λ indicates higher similarity. The substrings which are not present in the document will have a similarity of 0, which is the lowest possible similarity score. Thus each row of the table is the feature vector for the respective word, so in this higher dimensional space the document containing only the word "cat" would be represented as $\begin{bmatrix}\lambda^2 & \lambda^3 & \lambda^2 & 0 & \dots & 0\end{bmatrix}^T$. Using this, if one was to calculate the similarity/kernel between document 1 ("cat") and document 2 ("car") one would simply take the inner product of these two vectors: $K(\text{cat}, \text{car}) = \lambda^2 \cdot \lambda^2 + \lambda^3 \cdot 0 + \lambda^2 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot \lambda^3 + 0 \cdot \lambda^2 + 0 \cdot 0 = \lambda^2 \cdot \lambda^2 = \lambda^4$

In order to normalize this similarity one would divide it by the average similarity between the words themselves, i.e., $\frac{K(cat,cat)+K(car,car)}{2}$, which in this case

happens to be $K(cat, cat) = K(car, car) = (\lambda^2)^2 + (\lambda^3)^2 + (\lambda^2)^2 = 2\lambda^4 + \lambda^6$, and thus the final, normalized similarity becomes $\frac{\lambda^4}{2\lambda^4+\lambda^6} = \frac{1}{2+\lambda^2}$

Lodhi et al. [2002] also show another, slightly more comprehensive example without showing the calculations for different values of n, namely

$$K(\text{“science is organized knowledge”}, \text{“wisdom is organized life”}) \tag{2.7}$$

The decreasing values for this kernel, and values of $n = 1, 2, 3, 4, 5, 6$ are: $K_1 = 0.580, K_2 = 0.580, K_3 = 0.478, K_4 = 0.439, K_5 = 0.406, K_6 = 0.370$ which intuitively makes sense, as a lower value of $n = 1$ means that the two documents only need to contain the same characters, whereas with a value of 6 the two documents have to match to a much higher extent.

As one can imagine, performing all these computations becomes very costly, and the matrix becomes huge as the value of n increases and the documents become longer. In order to combat this, Lodhi et al. [2002] show how one can calculate these kernels very efficiently (actually linearly in the document's length) using dynamic programming.

2.2.4 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a method for finding the best linear combination of features which characterize or separate two or more classes. The idea of LDA is to take a set of data points in a feature space where the data points are not linearly separable, and project them *down* to a lower dimensional space where they are linearly separable. In order to do this, LDA maps the features into a new space where the distance between all the classes are maximized, while the scatter (variation) within each class is minimized – intuitively bringing all classes closer within themselves and as far away as possible from the other classes. More formally, given two different classes of data points with mean μ_1, μ_2 and standard deviation σ_1, σ_2 respectively, LDA aims to map the features into a lower dimensional space where the following term is maximized:

$$\frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \tag{2.8}$$

The intuition is that the numerator should be as large as possible, and the denominator as small as possible, effectively maximizing the distances between – and minimizing the variation within each class – as was the original goal. LDA also scales to higher dimensions, where one calculates a centroid between all classes, and aim to maximize the distance between the mean of each cluster and this centroid.

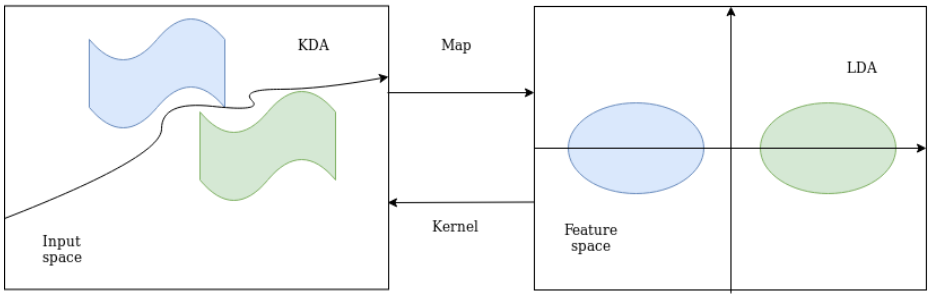

Figure 2.2: A visualization of Kernel Discriminant Analysis.

2.2.5 Kernel Discriminant Analysis

Kernel Discriminant Analysis (KDA), also known as Kernel Fisher Discriminant Analysis, is a kernelized version of LDA. This means that given the input space (where the classes are typically not linearly separable) one uses a kernel function to map them into a higher dimensional space where they (hopefully) are. Next, LDA is applied in this higher dimensional feature space in order to bring the features back into the original input space, but this time hopefully being linearly separable. A visualization of this process can be seen in figure 2.2.

2.3 Evaluation of Models

Evaluation of machine learning models is essential. If the model is to be used on real life tasks, a notion of how well the model is performing must be provided. Several metrics are used in order to evaluate and assess the performance of a machine learning model, some of which will be described below.

2.3.1 Accuracy

Accuracy is one of the simplest methods for measuring the performance of a system. The accuracy of the model will simply be the fraction of correct classifications it makes out of the total number of examples it is being tested on. That is:

$$accuracy = \frac{num\_correct}{total\_number\_of\_test\_examples} \tag{2.9}$$

Accuracy is a simple, yet effective measurement of how the model is performing over all.

2.3.2 Precision and Recall

Precision and recall are two highly correlated measures of relevance, and are frequently used in the field of Information Retrieval [Sasaki, 2007]. Precision is the fraction of relevant instances retrieved among all retrieved instances, while recall is the fraction of relevant instances retrieved out of all the relevant instances that exist. If the instances in question are documents, precision is how much of what the model has returned is actually relevant, and recall is how well the model has covered all the relevant documents. Both precision and recall are values between 0 and 1. A high recall value might be important when retrieving medical or legal documents, as the user needs as many relevant documents as possible, and can withstand quite a few irrelevant documents to obtain these. A high precision is desirable when the user only wants what is relevant, but perhaps does not need *all* the relevant documents.

The reason the two measurements are so intertwined is that if one wants to maximize the recall metric, one can simply have the model return all the documents (thus also returning all relevant documents and achieving a maximum recall value of 1), but the precision will probably be low since a lot of irrelevant documents are returned. On the other hand, if one wants to achieve a high precision score, one can have the model just return one document that it is absolutely sure about, which would make the precision 1 as all the documents retrieved are relevant. However, the recall value would be low, as only a small fraction of all relevant documents has been retrieved. Due to this, one might want to maximize both precision and recall at the same time. For this purpose, the F_1 score exists, which will be covered in the following section.

2.3.3 F1 score

The F_1 score is the harmonic mean of the precision and recall metrics [Sasaki, 2007]. It reaches its highest value of 1 when both precision and recall are perfect, and 0 if both precision and recall are zero. It is calculated as follows:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.10}$$

The macro-averaged F_1 score is calculated by first computing the F_1 score for each class, and then taking the average across all classes. For instance, each of these classes in NLI would be the L1. The idea is to favor more consistent performance across classes (L1s), rather than simply measuring global performance across

all samples. A high macro-averaged F_1 score therefore ensures that the model is performing well across all classes, regardless of the size of the classes. This because the average of a small class will have the same weighting as the average of a class with many samples, making the macro-averaged F_1 score suitable for data sets with class-imbalances.

2.3.4 Cross-Validation

Cross-validation comes in several different variants and is a technique for assessing how the results of a statistical analysis will generalize to an independent data set. Using cross-validation can help evaluate whether the model is overfitting, and to give an insight on how the model will generalize. This is often hard to do with only one test set as one might run into issues like bias and high variance depending on the selection of the test set.

K-fold Cross-Validation, combats this problem by dividing the training set into k parts, validating it on each of the k parts, training on the remaining $k-1$ parts. The overall performance can then be evaluated across the different folds, and see if the results are consistently good – that the model is consistently stable, and not just "lucky" on some parts. This ensures that all the data is used for training ($k-1$ times), and also being used for validation. This should reduce overall bias and variance, and be a testament to whether the model actually generalizes well, regardless of what part of the data it is being tested on. There are several ways of creating the folds in cross-validation. One technique is to randomly shuffle the data set before dividing it. However, there exist more sophisticated variations such as *stratified* cross-validation, where what goes into each separate fold is taken into consideration. For instance, on a binary data set where each sample is labeled either true or false, one might want to divide the folds so that each fold contains a roughly similar distribution of positive and negative samples.

2.4 Deep Learning

In order to understand the attention mechanism of deep learning, the basics of deep learning and sequence-to-sequence models must first be understood. The following section will cover the basics of deep learning and attention, as well as the advanced Transformer architecture which relies solely on attention.

2.4.1 Feed-Forward Neural Networks

Feed-Forward Neural Networks (FFNNs), also known as multilayer perceptrons, are a form of artificial neural networks. Neural nets draw their inspiration from

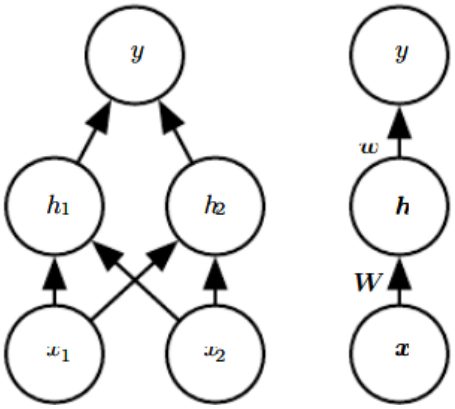

Figure 2.3: A typical feed-forward neural network composed of two layers. On the left, every unit of the network is drawn. On the right a more condensed drawing is composed of vectors representing each layers activation. Figure from Goodfellow et al. [2016] with permission from Ian Goodfellow.

biology and the brain, in the sense that a FFNN consists of nodes similar to the perceptrons of the human brain. Similarly to the brain, the perceptrons are connected via weighted edges, and sorted in layers. Each node typically has a weighted edge going out to every single node at the following layer, representing a vector-to-scalar function. The neural net starts at its input nodes where each node takes in a value of the input feature vector. These input values are then fed through to the network by multiplying the input by each output weight at that layer. The nodes in the next layer then take the weighted output of all incoming weights as their input, and repeat the same process forward. The layers of nodes which operate on the output of other layers, as opposed to processing the input or output, are called *hidden layers*. More formally, for each neuron j, the outputted value, v_j, is calculated as follows:

$$v_j = \sigma_j(\sum_{j'} w_{jj'} \cdot v_{j'}) \tag{2.11}$$

where $w_{jj'}$ is the weight *from* neuron j' *to* the neuron j, and σ_j is a non-linear function of neuron j [Lipton et al., 2015]. This process calculation propagates throughout all the hidden layers in the network until the final output layer is reached, and the final result is outputted. A visualization of a neural net can be seen in figure 2.3.

Taking a step back, the goal of a neural network is to approximate some func-

tion f^* based on training data. For a classifier, the function to optimize becomes $f^*(\mathbf{x}) = y$, where $\mathbf{x}$ is the input vector, and y is the target label. Neural nets are called networks because this function is typically approximated by chaining multiple functions together, e.g. $f^*(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}))$. $f^{(1)}$ is the first layer of the network, $f^{(2)}$ is the second layer of the network, and so on. The length of this chain of functions is called the *depth* of the network, giving deep learning its name. If all the functions of the chain are linear functions, the neural network itself will be a linear function. This is a problem, as the network will not be able to learn non-linear representations. For this reason, one ore more *activation functions* are typically applied to different parts of the chain. These activation functions are non linear, and are typically denoted as σ. Activation functions will be discussed in more detail in section 2.4.6.

Viewing the network as a whole, the weights of all neurons at each layer can be represented as vectors and matrices. As an example, a simple two-layered neural net can be described as

$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \sigma(\mathbf{W}^T \mathbf{x} + \mathbf{c}) + b \tag{2.12}$$

where $\mathbf{w}$ are the weights of the neurons of the second layer, $\mathbf{W}$ the weights of the first layer, and $\mathbf{c}$ and b are linear biases. As can be read from the equation, the two functions are still linear at their core, but the activation function, σ, allows the second linear layer to operate on the non-linear output of the first layer.

The final output of the network can be compared to an objective loss function, which outputs a single value indicating how far off the network was from the correct output. The largest difference between the linear models discussed in section 2.2 and neural networks, is that the non-linearity of a neural network causes most loss functions to become non-convex. This requires neural networks to be trained using iterative, gradient-based optimizers that merely drive the cost function to a very low value, as opposed to the linear equation solvers and optimization algorithms used in linear regression models and SVMs with global convergence guarantees. The training algorithm is almost always based on using the gradient to descend the cost function in one way or another [Goodfellow et al., 2016].

The learning of the network is performed by adjusting the weighted edges of the network. This is typically done by a technique called backpropagation [Svozil et al., 1997]. Briefly, backpropagation calculates the gradient of the loss function, $\mathcal{L}$, of the network with regards to each weight in the network, and adjust the weights in the direction that would reduce the loss proportional to each weight's contribution to the loss. The weights are typically then calculated using gradient decent [Lipton et al., 2015] or other optimization methods (see section 2.4.8).

Typically, the training of neural networks is performed in batches of training

samples. Instead of calculating and adjusting the weights according to the loss for each single training case, several training cases are sent through as batches. Within each batch, the gradients are calculated and kept track of for each training instance, but the actual updating of the weights is not performed until the entire batch has been processed. The weights are then simply updated using the accumulated gradients obtained in within the batch [Goodfellow et al., 2016]. Training in batches reduces the computations needed for training neural networks, as the backpropagation step is not a trivial task, and performing it less frequently reduces the number of computations required.

2.4.2 Recurrent Neural Networks

Recurrent neural network (RNNs) are a family of neural networks for processing sequential data, with input of the form of a sequence of values $x^{(1)}, \dots x^{(\tau)}$ Goodfellow et al. [2016]. While standard feed-forward neural networks required a fixed input, most recurrent networks can scale to much longer sequences than what would be practical for networks without sequence-based specialization. Depending on how the network is built, most recurrent networks can process sequences of variable length. To go from multi layer networks to recurrent networks, sharing parameters across different parts of a model is required. Parameter sharing makes it possible to extend and apply the model to examples of different lengths and generalize across them [Goodfellow et al., 2016].

RNNs operate on a sequence which contains vectors at different time steps, $\mathbf{x}^{(t)}$, with the time step index ranging from 1 to τ. In recurrent neural nets, parameter sharing is done via recursion, where each hidden state is dependent on the hidden state before it. This recursion is expressed by the following relationship:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta}) \tag{2.13}$$

where $\boldsymbol{\theta}$ are the parameters of the model, and $\mathbf{h}$ represents the hidden state vector.

Recurrent neural networks can be built in many different ways – essentially any neural net based on recursion can be considered a recurrent neural net. A simple RNN for classification can be built using the following update rules:

$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b} \tag{2.14}$$

$$\mathbf{h}^{(t)} = \sigma(\mathbf{a}^{(t)}) \tag{2.15}$$

$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c} \tag{2.16}$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)}) \tag{2.17}$$

where the input-to-hidden connections are parametrized by the weight matrix $\mathbf{U}$, hidden-to-hidden recurrent connections parametrized by the weight matrix

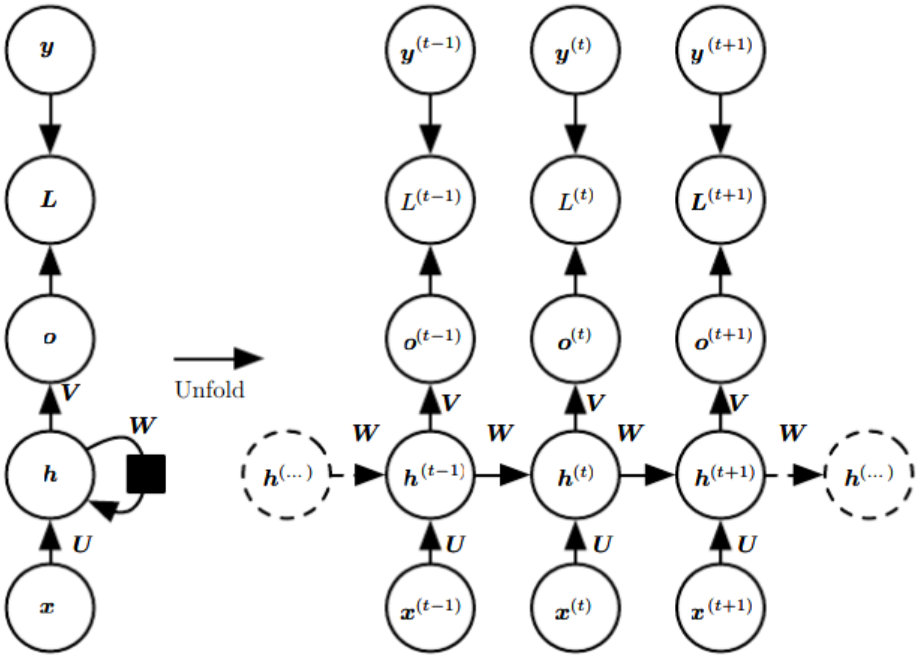

Figure 2.4: A typical recurrent neural network which produces an output at each time step. On the right, the network is unfolded and each node at each time step is drawn. On the left a more condensed drawing is composed using a black box to indicate the corresponding recursion. Figure from Goodfellow et al. [2016], with permission from Ian Goodfellow.

$\mathbf{W}$, and hidden-to-output connections are parametrized by the weight matrix $\mathbf{V}$. Using these rules, the network will start at the initial hidden state $\mathbf{h}^{(0)}$, typically initialized randomly, and for each time step update the next hidden state and output, based on the previous hidden state in the input at time step t. The softmax function will be covered in section 2.4.5. A visualisation of this particular recurrent neural net can be found in figure 2.4.

The final hidden state produced by the network can be viewed as a lossy summary of the input sequence, as it maps a sequence of arbitrary length, $\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \ldots \mathbf{x}^{(2)}, \mathbf{x}^{(1)}$, into a fixed length vector $\mathbf{h}^{(t)}$.

2.4.3 Layer Normalization

One challenge in neural networks is that each hidden layer is highly dependent on the outputs of the previous layers. This can create problems like vanishing or exploding gradients. In addition, training state-of-the-art, deep neural networks is computationally expensive. One way to reduce the training time and sensitivity of the network is to normalize the activities of the neurons.

Lei Ba et al. introduce layer normalization, a technique for normalizing the input at each layer in the network. Layer normalization uses the distribution of the summed input to a neuron over a single training case to compute mean and variance, which are then used to normalize the summed input to that neuron. Layer normalization computes this mean and variance from all of the summed inputs to the neurons. Lei Ba et al. also give each neuron its own adaptive bias and gain which are applied after the normalization, but before the non-linearity.

Empirically, layer normalization can substantially reduce the training time of neural networks, and works for recursive neural networks as well.

2.4.4 Cross-Entropy Loss

The cross-entropy loss function, also known as log loss, measures the performance of a classification model whose output is a probability value between 0 and 1 for each class. In the binary case, for the predicted probability p for the correct class $y \in \{0, 1\}$, the cross-entropy loss is given by:

$$\mathcal{L}_{ce}(y, p) = -(y \log_2(p) + (1 - y) \log_2(1 - p)) \tag{2.18}$$

The intuition behind log loss is that it penalizes the model when the model is correct in its prediction, but the predicted probability is not 1.0. The penalty is even higher when the predicted label is incorrect and the probability for this incorrect label is high.

For instance, if the model predicts a probability of $p = 0.9$ for when the correct binary class label is $y = 1$, the log loss will be $\mathcal{L}_{ce}(1, 0.9) = -(1 \cdot \log_2(0.9) + 0) = 0.152$. However, if the correct label was actually $y = 0$, the loss would be $\mathcal{L}_{ce}(0, 0.9) = -(0 + 1 \cdot \log_2(0.1)) = 3.322$. Thus the model is penalized more for predicting a high probability for the 0 class. The cross-entropy loss function can easily be extended to multiclass problems by applying the log loss function to the predicted probability for each class and and the correct label, and sum up the log losses for all classes.

2.4.5 The Softmax Function

In order to supply the cross-entropy loss function with the predicted probabilities of the model, the softmax function is often used to map each class output

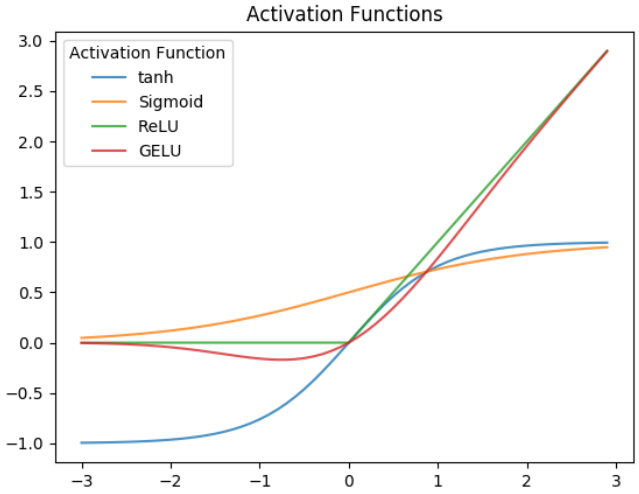

Figure 2.5: A simple plot showing the different output values for different input values for different activation functions.

to a probability. More formally, the softmax function is a mathematical function which takes a K-dimensional, real valued vector as input, and produces a probability distribution of K probabilities. If a model produces a vector of K-values, each value representing the value of a particular class, the softmax function maps this vector into a K dimensional vector where each value will be in the interval $(0, 1)$. The function also normalizes these values so that the sum of the probabilities will be 1. Given an input vector z of dimension K, the softmax function is given by

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}} \tag{2.19}$$

2.4.6 Activation Functions

As mentioned in the introduction to neural networks, an activation function is often applied to neurons. This activation function is typically a non-linear function, applied in order to enable the network to learn non-linear mappings. This subsection will cover the relevant activation functions for this Thesis. A plot of the different activation functions can be found in figure 2.5.

Sigmoid Function

The Sigmoid activation function is a simple function which maps real values into the interval (0, 1). The function is inspired by biology, where neurons are more stimulated the higher the input value. The Sigmoid function is given by:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.20}$$

As the Sigmoid function maps values into the (0, 1) range, it can be a useful function for predicting probability based output and has been applied successfully in binary classification problems, modeling logistic regression tasks as well as other neural network domains [Nwankpa et al., 2018].

A problem with the Sigmoid function however, is that negative output values will often be mapped closed to zero. This causes major drawbacks and problems such as gradient saturation, slow convergence and non-zero centred output. This can cause the gradient updates to propagate in different directions [Nwankpa et al., 2018].

Hyperbolic Tangent

The hyperbolic tangent function, or tanh, is similar to the Sigmoid function, but maps all output values to the interval (-1, 1). The tanh function is given by

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.21}$$

The tanh function combats the problem the Sigmoid function has with negative values, as large negative values will be mapped close to -1 instead of 0, and has been shown to give better training performance in multi-layered networks when compared to the Sigmoid function. The tanh function has mostly been used in NLP tasks and speech recognition tasks [Nwankpa et al., 2018].

Rectified Linear Unit – ReLU

A property of the tanh function is that it can only attain a gradient of 1 when the input is 0. This makes the tanh function produce some dead neurons during computation. A dead neuron is a condition where the activation weight is rarely used because the gradient becomes zero. This dead neuron limitation of the tanh function spurred further research in activation functions to resolve the problem, which as a result gave birth to the rectified linear unit (ReLU) activation function. As of 2018, the ReLU activation function is the most widely used activation

function for deep learning applications [Nwankpa et al., 2018]. The function is given by:

$$f(x) = max(0, x) \tag{2.22}$$

The ReLU function is both faster and offers better performance and generalization in deep learning compared to the Sigmoid and tanh activation functions [Nwankpa et al., 2018].

Gaussian Error Linear Unit – GELU

The Gaussian error linear unit, of GELU function, is a more novel, high-performing activation function for neural nets [Hendrycks and Gimpel, 2016]. While the ReLU activation function deterministically multiplies the input by 0 or 1, the GELU function determine this zero-one mask stochastically and dependent upon the input. Specifically, the GELU function is given by

$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x) \tag{2.23}$$

where $\Phi(x) = P(X \leq x)$, and $X \sim \mathcal{N}(0, 1)$ is the cumulative distribution function of the standard normal distribution. In this setting, inputs have a higher probability of being "dropped" as x decreases, so the transformation applied to x is stochastic, yet dependent on the input [Hendrycks and Gimpel, 2016]. Hendrycks and Gimpel [2016] estimate the GELU function as:

$$\text{GELU}(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3))) \tag{2.24}$$

Hendrycks and Gimpel show that the GELU function beats the popular ReLU function.

2.4.7 Regularization

Regularization techniques in deep learning is a collection of method for reducing over-fitting in neural networks. Over-fitting is simply when the model learns the training data *too* well, while not being able to generalize and perform well on the validation or test data. The most common techniques for achieving regularization is to apply some sort of penalty to the loss function of the neural net. This subsection will cover the relevant regularization techniques used in this Thesis.

Dropout

The dropout technique, as the name suggests, is to "drop" some neurons in the network at each training step – i.e. setting their contribution to 0 [Srivastava

et al., 2014]. Which neurons to drop is chosen at random each iteration with a probability p, typically around the 0.1 to 0.2 range. Without dropout, neurons can develop co-dependencies amongst each other during training, which limits the individual power of each neuron, and can lead to over-fitting on the training data. Dropout is applied in order to combat this effect, and has proven to be a powerful technique for improving test accuracy in neural networks.

2.4.8 Learning Rate and the Adam Optimizer

Typically, neural networks use Stochastic Gradient Decent or other optimization algorithms to minimize the empirical loss on the training data. Gradient decent does this by minimizing the objective loss function of a neural network. At each training step, the gradient of each weight of the model with regards to its contribution to the total loss is calculated, and each weight is adjusted accordingly in the opposite direction of the gradient. In other words, gradient decent follows the direction of the slope of the surface created by the objective function downhill [Ruder, 2016]. The learning rate, often denoted as α, indicates how large steps the model should take when descending the loss-landscape, and is multiplied by each gradient step. A large learning rate will help the model quickly reduce the loss during the first steps of training. However, taking large steps will make the decent coarse, and can lead to the model to getting stuck in local minima. Stochastic gradient decent has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another. These ravines are common around local optima, and causes oscillations across the slopes of the ravine while only making hesitant progress along the bottom towards the local minimum [Ruder, 2016]. A smaller learning rate can help mitigate the trouble of navigating these ravines, as the model makes smaller steps within. Using a smaller learning rate (typically values in the range between 0.01 and 0.0001, or even smaller) will allow the model to search the loss-surface in a more granular fashion. A smaller learning rate will make the model converge more slowly, but provides a more thorough search of the loss-landscape, potentially leading to better solutions.

The simplest form of learning rate is a simple constant which is multiplied by all gradients. Other approaches include reducing the learning rate over time. This helps the model reduce loss quickly in the beginning of training, and perform a more granular search as it reaches lower surfaces of the loss search space. Over time, more sophisticated methods for optimization have emerged. Next, the most popular optimization scheme will be covered, namely Adam.

Adam

Adaptive Moment Estimation (Adam) is a optimization scheme similar to gradient decent, which computes adaptive learning rates for each parameter of the model, based on the first and second moments of the gradients [Ruder, 2016]. Keeping track of the momentum of the gradients can help mitigate the ravine-problem discussed above [Ruder, 2016]. In Adam, the first moments at time t, m_t, are the exponentially decaying average of past gradients, and the second moment, v_t, is the exponentially decaying average of past squared gradients. More formally, the moment vectors m_t and v_t are calculated as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.25}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.26}$$

where β_1 and β_2 are hyper-parameters with recommended values of 0.9 and 0.999 respectively, and g_t is the vector of gradients. As the gradient moments are all zero initially, Kingma and Ba observe that the moment vectors are typically biased towards zero. In order to counteract this bias, the moment estimates are bias-corrected as follows:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1} \tag{2.27}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2} \tag{2.28}$$

Finally, the model parameters, θ, are updated using the following equation:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.29}$$

where α is the learning rate, and ϵ is a very small constant used for avoiding division by zero – typically 10^{-8}. Adam is one of the most popular optimization schemes in deep learning, and is empirically shown to work well in practice and compared favorably to other adaptive learning-method algorithms [Ruder, 2016]

2.4.9 Sequence-to-Sequence Models

In many NLP tasks the goal is to generate one sequence based on another sequence. An example is machine translation, where the goal is to generate a translated sentence $T2$, based on the input sentence $T1$ of a different language. Both sentences can be viewed as a sequence of words. A common architecture to for solving this is sequence-to-sequence (seq2seq) modeling [Sutskever et al.,

2014]. A seq2seq architecture consists of two modules: an encoder and a decoder. The encoder is responsible for encoding the input sentence, $T1$, and producing an encoded output representation of the input. The decoder then takes in this encoded representation of the input and generates an output sequence based on it. This can for instance be done by using a RNN as the encoder, and another RNN as the decoder. In machine translation, the encoder-RNN would take in a vector representation of the input sentence, and feed it through itself, generating a new hidden state for each input word based on the word and the previous hidden state. When the encoder-RNN is done processing the input sequence, it will have a final internal hidden state. This internal hidden state is then fed in as input to the decoder-RNN. The decoder-RNN uses this hidden state as its initial hidden state, as well as a start-of-sentence token as input to start generating the first element of the output sequence (i.e one translated word). This process continues until the decoder-RNN produces the end-of-sentence token, which indicates that the full output sentence has been generated.

Trying to solve the same task with a single RNN would imply that the RNN would have to produce one output token for each input token. Using a seq2seq architecture enables the input- and output sequence to be of different lengths, as the only input the encoder requires from the encoder is its final hidden state obtained after processing the input sequence. Additionally, this architecture (hopefully) enables the encoder to capture information from the entire input sequence and make it available to the decoder, as opposed to having an RNN-produce the next output-token based only on the previous input tokens. However, there are some caveats to this approach, which will be discussed in the next subsection.

2.4.10 Attention

As discussed above, a sequence-to-sequence architecture enables the encoder model to encode the entire input sequence into one, encoded hidden state representation. This is a potential bottleneck for the performance of the system [Bahdanau et al., 2014]. A problem is that the first words of the input sequence become less pertinent, as they will have less impact on the final hidden state – especially for longer inputs. Often, one of the first tokens of the input sentences might be important for one of the later tokens of the output tokens. For instance, when translating from English to German, a verb which appears early in an English sentence is often put at the end of a German sentence, creating a great spatial difference between the two tokens, even though they are highly related. A method for solving this problem is to use an attention mechanism.

The basic idea behind the attention mechanism is to not only feed the decoder the final hidden output state of the encoder, but *all* of the hidden states the encoder produces when processing the input sequence. By letting the decoder

have an attention mechanism, the encoder is relieved from the burden of having to encode all information in the source sentence into a fixed length vector [Bahdanau et al., 2014]. Making all the hidden states available to the decoder enables it to weigh each of the hidden states individually, thus deciding which parts of the input sequence it wants to "pay attention to". These weights are learned, often using a feed-forward neural network within the decoder. After the decoder has weighed each hidden state, the hidden states are run through the softmax function, and one single vector is created where each hidden state is multiplied by its softmax score. The decoder then calculates what is called the *context vector*, which consists of the weighted sum of all the hidden state vectors multiplied by their softmax score. This vector is then concatenated with the hidden state that the encoder produces after processing the first start-of-sentence token. This concatenated vector is finally fed thorough a FFNN, which then produces the first output token. This output token is then fed into the decoder again, which produces a new hidden state, and the same process continues until the end-of-sentence token is produced.

Self-Attention

The attention mechanism allows the decoder to attend to different parts of the input sequence by looking at the individual hidden states produced by the encoder at each step. However, the encoder itself does not have any attention mechanism. In order for the encoder to create even more meaningful representations of the input sequence, "self-attention" can be applied within the encoder. Self-attention allows the encoder to attend to all words in the input sequence for each word in the input sequence. That is, the self-attention mechanism relates different position of a single sequence in order to compute a representation of the sequence [Vaswani et al., 2017]. For example, given the input sequence "The animal didn't cross the street because it was too tired", the word *it* refers to the beginning of the sentence, namely "the animal". A standard RNN will be able to capture this to some extent via its hidden state, but by giving the encoder an internal attention mechanism it can pay more attention to specific words explicitly.

Self-attention is implemented using three weight matrices called the *query, key* and *value* matrices, which are learned during training. For each word-embedding in the input sequence, the word-representation is multiplied by the query weight matrix in order to obtain a query vector representation of the current word. This word, q_n, will be the word being attended to at position n in the sequence. Then, for all word-embeddings in the sequence, a key vector is calculated by multiplying the vector representation of the current word with the key weight matrix. Similarly, the value vector for each word is obtained by taking the dot product of the word and the value matrix. Next, the query vector q_1, the word which the model is currently attending to, is multiplied by all key vectors in the

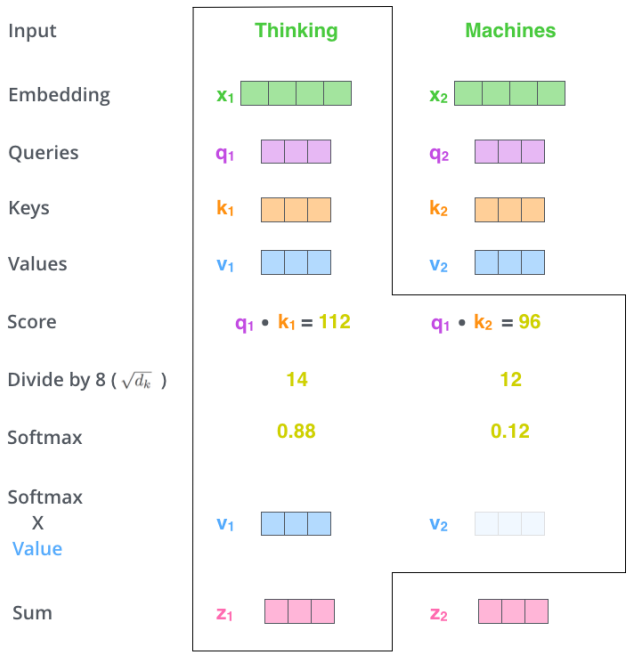

Figure 2.6: An example of self-attention for the input sequence "Thinking Machines". Figure from Alammar [2018], with permission from Jay Alammar.

sequence $k_1 \dots k_n$ to obtain an attention score for each word. These scores are then softmaxed, and multiplied by the corresponding value vector. The intuition is that relevant words which the model should attend to will receive a high score, while drowning out words which are irrelevant. Next, the softmax-weighted value vectors are summed up to give a final, attention-encoded representation of the input sequence at this position.

A visual dummy example of this process can be seen in figure 2.6. The example shows how the self-attention encoded representation (z_1) of the word "Thinking" in the sequence "Thinking Machines" would have been calculated using dummy values. The fifth step of dividing by $\sqrt{d_k}$ is specific to the Transformer architecture, which will be discussed in the following section.

2.4.11 Transformers

Vaswani et al. [2017] proposed a new architecture called the Transformer, which

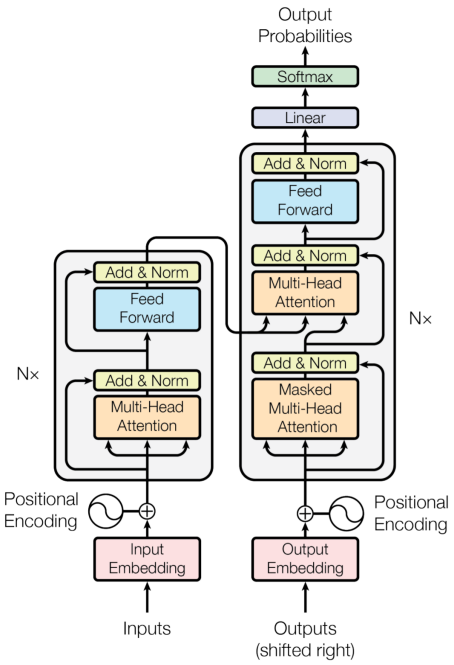

Figure 2.7: A visualization of the Transformer architecture. Figure from Vaswani et al. [2017], with permission from Jakob Uszkoreit and Aidan Gomez.

relies solely on attention mechanisms. As the Transformer does not use recurrence nor convolution, most of the required calculations can be carried out in parallel while still having superior quality compared to previous encoder-decoder architectures [Vaswani et al., 2017]. A visual overview of the transformer architecture can be found i figure 2.7. As can be seen in the figure, the transformer is an architecture composed of several sub-modules, each of which will be described in more detail below.

The overall architecture of the Transformer is conceptually simple. The model consists of N encoders and decoders, denoted by "Nx" in the figure. Each encoder consists of a multi-head attention encoding layer as well as a feed-forward linear layer. The multi-head attention layer will be described in more detail in the following subsection, but for now it is sufficient to know that it uses self-attention to create and encoded representation of the input sequence. After the multi-head-attention layer has encoded the input, it is added to the original input. This can be seen through a residual connection going around the multi-head attention

layer. After the original and the encoded input has been added together, the aggregated input is then normalized using layer normalization. This process is illustrated by the "Add & Norm" box. This aggregated input is then fed into a linear feed-forward layer, whos output again is added and normalized in the same fashion as the output of the multi-head attention layer. This describes *one* encoder. Multiple instances of this encoder architecture are then stacked N times, where the next encoder takes the output of the previous encoder as input.

The stack of decoders is similar to the encoders, but with a few altered details. The first difference is that the first multi-head attention layer is *masked*. This multi-head attention layer processes the output embeddings, which are produced by the model itself, starting with the start-of-sentence token. The term masking simply means that attention values for words which have not been produced yet are set to zero, which means that the decoder can only attend to words it has already produced.

The second difference is that the second multi-head attention layer processes the aggregated and normalized output of the first masked multi-head attention layer, as well as the outputs of the encoder stack. This means that the entire input sequence is first encoded by the encoders, and is then used as a "static" input for all the decoders while the decoder stack processes the output sequence in an auto regressive manner (meaning that each output token produced is fed back as input while processing the next token). At the end of the decoder stack there is a linear layer which maps the real valued output from the decoder to a vector of a desired size. The size of this output vector can for instance be the number of classes, if the Transformer is used for a classification task, or be the same size as the output vocabulary if used for machine translation or other sequences. This final output is then fed through the softmax function in order to produce a probability for each class, or the probability of each word in the target vocabulary.

Multi-Head Attention

The multi-head attention layer used in the Transformer architecture is simply multiple instances of self-attention stacked on top of each other. This means that each attention "head" has its own query, key and value matrices, which is initialized randomly. This allows each head to project the input into different representation sub-spaces, allowing for more complex attention. Ideally, this would imply that one head perhaps learns to pay specific attention to, say, the pronouns in the input sentence, while another head perhaps pays attention to nouns.

The actual implementation of the Transformer is carried out using matrices for the aforementioned vectors and the self-attention vectors. This means that each the input, query, key and value vectors are all represented as matrices for

more efficient calculations. Based on this, the self-attention steps described in 2.4.10 can be condensed into one simple calculation:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.30}$$

Vaswani et al. hypothesise that, as the dot products grow large in magnitude, they can push the softmax function into regions where it has extremely small gradients. To counteract this effect, each output is scaled by a normalization constant $\sqrt{d_k}$ – the square root of the hidden dimension of the transformer.

Positional Encodings

As the Transformer contains no recurrence nor convolution, the model will not be able to make use of the ordering of the sequence. To fix this, some information about the relative or absolute position of each token in the sequence must be injected. Vaswani et al. solve this by adding positional encodings to the input and output embeddings. This is done by using sine and cosine functions of different frequencies:

$$\text{PE}(pos, 2i) = \sin(pos/10000^{2i/d_{model}}) \tag{2.31}$$

$$\text{PE}(pos, 2i+1) = \cos(pos/10000^{2i/d_{model}}) \tag{2.32}$$

where *pos* is the position of the word in the sequence, and i is the dimension of the positional encodings (which will be the same dimension as the word embeddings, for easy summation of the vectors). That is, each dimension of the positional encoding corresponds to a sinusoid.

2.5 Natural Language Processing

Natural language processing (NLP) is the field of programming computers to process and analyze large amounts of natural language data. As NLI is a subfield of NLP, it will be natural to cover the basics and the relevant features of NLP. The following sections will discuss some traditional statistical features and word representations of NLP, as well as more recent deep learning approaches.

2.5.1 Lemma and Function Words

When discussing NLI features, and NLP features in general, two central concepts are lemmas and function words. A lemma is a word that one would find in a dictionary, i.e., the root form of the word. For instance, *run, runs, ran* and

running are forms of the same lexeme, and *run* is the lemma. Lemmas are for instance very useful for search and querying, where one might want to return a document which contains the same lemma, even though the query was in a different tense than how it appears in the document.

Function words are words which contribute to the syntax rather than the meaning or semantics of a sentence, such as "the" and "a", etc. Typical function word classes include articles, pronouns and conjunctions. In light of NLI, the use of function words can be an indicator of how an author composes sentences and the stylistic properties of their writing. In addition, function words are topic independent, as they only carry information about *how* the author writes, and not *what* the author is writing about.

2.5.2 N-grams

The construction of n-grams is an essential technique for text processing, and is defined as a contiguous sequence of n items from a given sample of text or speech. n-grams are typically constructed at either the word or character level. An example of a word-level n-gram constructed from the sentence "the dog ate the cat" would be every single word by itself when $n = 1$ i.e., "the", "dog", "ate" ... and so on. This is called a bag-of-words model, since the order of the words is irrelevant, and it only models that a word has appeared, not what words it appeared after. With $n = 2$, each word is concatenated with the word before it, i.e., "the dog", "dog ate", "ate the", "the cat". n-grams where $n > 1$ are used to capture the context a word or character appears in. An example of n-grams for different values of n and the sentence "the dog ate the cat" can be found in table 2.2.

n	Character level n-grams	Word level n-grams
1	t, h, e, , d, o, g, , a, t, e, ...	the, dog, ate, the, cat
2	th, he, e , d, do, og, g , a, ...	the dog, dog ate, ate the, the cat
3	the, he , e d, do, dog, og , ...	the dog ate, dog ate the, ate the cat
4	the , he d, e do, dog, dog , ...	the dog ate the, dog ate the cat
5	the d, he do, e dog, dog , ...	the dog ate the cat

Table 2.2: Comma separated character and word level n-grams for different values of n, given the sentence "the dog ate the cat".

2.5.3 Part-of-Speech Tagging

Part-of-Speech (POS) tagging is the process of marking a word in a text as corresponding to a particular part of speech, based on both its definition and its context. That is, its relationship with adjacent and related words in the context it appears in [Voutilainen, 2003]. Typically this includes tagging/classifying each and every word with a predefined set of POS tags, i.e., "N" for "Noun", "NP" for "Noun Phrase", "VP" for "Verb Phrase", "D" for "Determiner" and so on. The details of how to obtain such POS tags automatically will not be covered, but typically they are deduced using supervised learning with training sets annotated by humans. POS tags help disambiguate words, for instance by tagging words like "milk" with either "V" or "N", disambiguating whether it is meant as milk in the noun form, or in its verb form "to milk something".

2.5.4 WordPiece Tokenization

WordPiece tokenization is a technique for building vocabularies which are meaningful to machine learning models from sub-word units. WordPiece tokenziation is completely data-driven and guaranteed to generate a deterministic segmentation for any possible sequence of characters. The WordPiece model is generated using a data-driven approach of training a language model to maximize the models likelihood of the training data, given an evolving vocabulary. Wu et al. deploy a greedy algorithm and find that their models perform better using WordPieces – possibly because the WordPieces provide an essentially infinite vocabulary without resorting to characters only.

Wu et al. [2016] provide an example of a word sequence mapped to a WordPiece sequence to illustrate how such learned sub-word units of a particular sentence might look:

- Jet makers feud over seat width with big orders at stake
- _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

Here, the model has larned a vocabulary where the word "Jet" has been broken into two WordPieces, "_J" and "et", and the word "feud" has been broken into "_fe" and "ud". "_" is a special character added to mark the beginning of a word.

2.5.5 Word Embeddings

Word embeddings are continuous vector representations of words [Mikolov et al., 2013a]. Encoding words as vectors allows for mathematical operations on words, and using high quality word embeddings usually increases performance over using more traditional word representations such as simple n-grams [Mikolov et al.,

2013a]. For instance, after learning word embeddings, Mikolov et al. [2013b] report that the resulting vector after performing the calculation vec("Madrid") - vec("Spain") + vec("France") is closer to vec("Paris") than any other vector in the vocabulary.

Most of today's word embedding approaches are based on using neural networks to learn the encoded real-valued vector representations of single words. Two of the simplest approaches are the bag-of-words (BOW) model and the word skip-gram model, often referred to as word2vec models. The BOW model simply trains a shallow neural network to predict a single word based on a context window of surrounding words, where the words are typically one-hot encoded. After training a neural language model in this fashion, the weights of the hidden layer represent a vector space where each vector corresponds to a specific word in the vocabulary. These vectors can then be used as word embeddings. The word skip-gram model reverses the task of the BOW model, and predicts context words based on the a single input word. Other neural networks such as RNNs have also been successfully applied for learning word embeddings [Mikolov et al., 2013b].

Chapter 3

Related Work and Motivation

As the main goal of this Master's Thesis is to explore how attention-based systems can be used to improve performance on the task of NLI, two main fields of literature will be relevant: the general field of NLI research, and research conducted on attention-based systems. In light of this, the literature review section has been separated into two sections, namely NLI literature and attention literature. In addition, a meta-section is included, which will aim to cover how the literature review search was performed. Finally, the chapter is ended with a motivation section which sums up the key takeaways of the literature review, and relates these to the initial Research Questions, and the experiments which will be carried out in the following chapter.

3.1 Literature Review

For attention related literature, most of the basics have been covered in the background theory section. This includes the ubiquitous Transformer architecture which relies solely on attention. As the Thesis started as a curiosity for BERT – a Transformer based, deep learning architecture – and its possible application for NLI, the attention literature review was accordingly performed in a top-down approach: starting at the top with the recent Devlin et al. [2018] article, reading the work which this article points to, carrying on all the way down to Transformers, attention and sequence-to-sequence models.

As for the field of NLI, a general search was first conducted, based on pointers from the Thesis' Supervisor. The initial search showed that much of the work in

the field of NLI is summed up in two shared NLI tasks – one in 2013 and one in 2017. Both of these shared tasks contain results and comparisons of 29 and 19 different approaches to solve the task of NLI, respectively – a good starting point. Similarly to the attention literature search, a top-down approach was applied. The search started at these two shared tasks, with emphasis on the more recent 2017 shared task. In practice, this includes considering the top contenders of each shared task, exploring their approaches, then reading the relevant work which these articles point to, and so on. As one of the Research Questions of this Thesis is how attention-based systems can be used in combination with current state-of-the-art approaches to improve performance on the task of NLI, the main emphasis have been on the work which has given good results in forms of accuracy or F_1 scores. Most of the NLI literature share similar features, and much of the work conducted during the 2017 shared task points back to the 2013 shared task. As for NLI literature post 2017, a new search was carried out. This search was simple, and involved querying the Google- and Google Scholar search engines for "Native Language-Identification", filtering out articles preceding 2018. Work from pre 2013 has mostly been ignored as most of the successful approaches were made during the two shared tasks. Most of the literature read has only been concerned with NLI where English is the second language, as this is the language for which there exists the most labeled data.

3.2 Native-Language Identification

The field of NLI is in a broader perspective quite young, and only gained serious momentum during the previous decade. Most notably the field gained traction after the work of Koppel et al. [2005], which according to Tetreault et al. [2013] was the first in the field. Koppel et al. trained an SVM model on a vast amount of features including POS tags, n-grams and grammatical errors on the ICLE data set [Brooke and Hirst, 2013]. Since then there have been two major events in the field of NLI, namely two shared tasks – the first one in 2013 and the second in 2017. The first task held in 2013 had 29 teams from all over the world participating, making it one of the biggest NLP competitions that year alone [Malmasi et al., 2017]. The winning team achieved an accuracy of 0.836 on the given test set. The second shared task of 2017 had 19 teams participating using both written and spoken responses, with a winning accuracy of 0.932 using both text and speech data, and an 0.882 accuracy when given essays only. These two shared tasks helped unify the community and give a common platform for evaluating and comparing different NLI systems.

3.2.1 The 2013 NLI Shared Task

As NLI gained more traction as a field, one of the main problems up until 2013 was that the field was lacking unification. There was no collective way of evaluating different NLI systems, as many of the publications before 2013 had used different sets of L1s, as well as different evaluation metrics. In addition, there was a lack of data suitable for the task of NLI. As the field gained more interest as a sub-task of NLP, the 2013 NLI Shared Task was held in order to address these problems. The solution to the above mentioned problems was the creation of the TOEFL11 corpus [Blanchard et al., 2013]. This data set did not only bring more data to the field, but provided a common set of languages to use for the task, as well as evaluation standards that everyone would use, providing a common ground for comparison of the different systems. The TOEFL11 data set will be described in more detail in section 3.2.2.

The 2013 NLI shared task was divided into three sub-tasks, namely Closed-Training, Open-Training-1 and Open-Training-2. Closed-Training was the main task and the one that received the most submissions. This task consisted of only using the training and development set, and nothing else. Conversely the Open-Training-1 task allowed the use of any training data *excluding* data from TOEFL11. The final task, Open-Training-2 allowed use of *any* data, including the TOEFL11 data set itself. However, the Open-Training-1 showed that training on external corpra while being tested on the TOEFL11 test set caused a significant drop in accuracy. For instance, the winners of the Open-Training-1 task achieved an accuracy of 0.565, as opposed to their 0.802 accuracy on the closed task. When being evaluated using 10-fold cross-validation the winning team achieved an accuracy of 0.846 using SVMs. Additionally, an observation made during the 2013 task was that typically the teams that performed well with regards to accuracy on the test set, also performed well when being cross-validated [Tetreault et al., 2013].

The approaches used in the 2013 shared task will only be covered briefly, as most of the best performing approaches were refined and used for the more recent 2017 shared task, with better results. These will be covered in greater detail in section 3.2.4.

The overwhelming majority of teams participating in the 2013 shared task used SVMs. Tetreault et al. [2012] additionally showed that performance could be increased by using ensemble methods for combining classifiers. The most common features used during the 2013 shared task were word, character and POS n-grams. Four of the top five teams used at least word 4-grams, and some as high as 7 and 9 [Malmasi et al., 2017].

3.2.2 TOEFL11

The TOEFL11 data set [Blanchard et al., 2013] consists of English essays written by people with different first languages for a college-entrance test, and stands for the Test of English as a Foreign Language. The data set was the first data set created specifically for NLI. The corpus contains 1,100 essays per language sampled, which is aimed to be distributed as evenly as possible across 8 different tasks/topics. Each essay is also labeled with the score it got, ranging from low, medium or high. The 11 languages included in the corpus are Arabic (ARA), Chinese (CHI), French (FRE), German (GER), Hindi (HIN), Italian (ITA), Japanese (JAP), Korean (KOR), Spanish (SPA), Telugu (TEL) and Turkish (TUR). A full overview of the languages as well as how many essays per prompt (topic), per language available in the corpus contains can be found in table 3.1. During the 2013 shared task the TOEFL11 data set was split into three parts: one training part consisting of 900 essays per L1, one development set consisting of 100 essays per L1, and the remaining 100 essays were put into the test set [Blanchard et al., 2013].

Language	P1	P2	P3	P4	P5	P6	P7	P8
Arabic	138	137	138	139	136	133	138	141
Chinese	140	141	126	140	134	141	139	139
French	158	160	87	156	160	68	151	160
German	155	154	157	151	150	28	152	153
Hindi	161	162	163	86	156	53	158	161
Italian	173	89	138	187	187	12	173	141
Japanese	116	142	140	138	138	142	141	143
Korean	140	133	136	128	137	142	141	143
Spanish	141	133	54	159	134	157	160	162
Telugu	165	166	167	55	169	41	166	171
Turkish	169	145	90	170	147	43	167	169
Total	1,656	1,562	1,369	1,509	1,648	960	1,686	1,683

Table 3.1: The number of essays per language per prompt/topic featured in the TOEFL11 data set. Figure from [Blanchard et al., 2013]

3.2.3 Between Shared Tasks

In between the 2013 shared task and the 2017 one, there was another related task focusing on spoken responses in 2016, called the Computational Paralinguistics Challenge [Malmasi et al., 2017]. This challenge was designed to explore speech-

based NLI in more detail. The data set provided contained 64 hours of speech from 5,132 non-native speakers of English. This results in roughly 45 seconds per speaker, which represented the same L1 languages as the TOEFL11 data set from 2013. Unfortunately it was not possible to distribute the raw speech data, so the data set consisted of textual transcripts as well as speech vectors called i-vectors. The winning team achieved an accuracy of 0.813 on this speech data alone [Malmasi et al., 2017]. It is sufficient to know that an i-vector is a vector of fixed length (in this case 800), which has been created as a lower-dimensional representation of high-dimensional sequential recordings of speech data. They were originally created for use in speaker recognition, and later used for language recognition.

Another important event in between the two tasks was that Ionescu et al. [2014] beat the previous best results from the 2013 task using string kernels. However, as these results were further improved using similar techniques in a submission for the 2017 shared task, the approach used will be covered in the following section.

3.2.4 The 2017 NLI Shared Task

The 2017 shared task was similar to the 2013 task, but aimed to utilize the new speech data available in combination with text for NLI. With 19 participating teams, the task was divided into three tracks: NLI for essay only (i.e., the same as the 2013 task), NLI for the spoken response only (which would be the same as the CPC speech challenge from 2016 mentioned above), and the new contribution of a combined task using both text and speech, dubbed the fusion track. Submissions in the fusion track showed that combining written and spoken responses provides a large boost in prediction accuracy for NLI systems [Malmasi et al., 2017]. Another trend from the 2017 shared task was that ensemble-based systems were again shown to be the most effective in all tasks.

Malmasi et al. also state that another motivation for the 2017 shared task was the rapid growth of deep learning methods for NLP during the time slot between the two tasks. Despite this, typically a lot of the same features were used in 2017 as were used in 2013, and SVMs were still the most popular approach. Just like in 2013, there were closed and open versions of the different tasks regarding the use of external data. However, possibly due to the significant drop in accuracy seen in the 2013 shared task, there were no submissions in the open competition. In contrast to 2013, the 2017 competition did not only use accuracy as its evaluation measure, but the official evaluation metric was macro-averaged F_1 score. All the participating groups were placed into tiers based on how well they performed. Each tier was statistically significantly different from the other tiers, while the groups within the same tier were calculated to be statistically similar, despite

having slightly different scores.

There were some primary trends observed in the 2017 shared task for NLI. First of all, again similarly to the 2013 task, most results showed that multiple classifier systems were effective, i.e. ensembles and meta-classifiers. Almost all of the top ranking teams employed some type of multi-classifier system [Malmasi et al., 2017]. Next, lexical n-grams were empirically the best single feature type. Evidence from various participants suggested that high-order character n-grams, as high as $n = 10$, were extremely useful for the task. Character n-grams were assumed to be especially efficient as they capture sub-word (morphological) information, as well as dependencies between words. A slight performance boost could also be gained by using syntactic features. Furthermore, feature weighting schemes were important. Many of the top teams applied some form of feature weighting, typically TF-IDF feature weighting. Regarding the speech-based systems, acoustic features seemed to be highly informative. The fusion of textual and speech features gave the overall best results. Furthermore, contrary to the initial motivation of the 2017 shared task, traditional classifier models continued to dominate over deep learning models. As will be discussed below, Ircing et al. [2017], as well as others, assume that this was due to the size of the TOEFL11 data set, and that more training examples could help deep learning models perform better.

In the following sections, some of the specific submissions for the 2017 shared task will be covered in detail, as these approaches are still prevalent in the current state-of-the-art of the field, and therefore relevant for answering Research Questions 1 and 3. The teams UnibucKernel and CEMI both placed in the top 1 tier on the fusion-track, making them both state-of-the-art on the fusion-track for the TOEFL11 data set in 2017. In addition, ItaliaNLP – the winners of the essay-only track – will be described, as it is the best performing system on the TOEFL11 data set so far.

3.2.5 UnibucKernel – 2017 Shared Task Winners

UnibicKernel [Ionescu and Popescu, 2017] were the overall winners of the fusion track, and also won the speech-only track with a top F_1 score of 0.932. The solution also achieved an F_1 score of 0.867 in the closed essay track, which placed the UnibicKernel in the top tier in this category as well – making the system statistically as good as the winners, ItaliaNLP.

The results of Ionescu and Popescu were achieved by extending the previous approach, found in Ionescu et al. [2014], using multiple kernel learning and kernel discriminant analysis. In the initial experiments on the development set, the team tried both KDA and another binary classifier called Kernel Ridge Regression (KRR). However, KDA beat KRR in all cases, which was consistent with the

previous results of Ionescu et al. [2014]. For this reason, KDA was used for the remaining experiments. Furthermore, three different kernels were experimented with: a character n-gram presence kernel, an intersection string kernel, as well as the RBF kernel. In addition, squared versions of each kernel were also included, as well as different combinations of all them. Only the best performing kernel will be covered, namely the first mentioned character n-gram presence bit kernel. This character n-gram presence bit kernel is defined as follows:

$$k_n^{0/1}(s,t) = \sum_{v \in \sum n} \text{in}_v(s) \cdot \text{in}_v(t) \tag{3.1}$$

where s and t are strings (or documents), present in an alphabet $\sum$, that is $s,t \in \sum^*$, and the function $\text{in}_v(x)$ is 1 if string v occurs as a sub-string in x, and 0 otherwise. Basically what this kernel function does is to define the similarity between two strings s,t as the binary sum of all n-grams which are present in both s and t, for all possible n-grams. In other words, this kernel is a feature map that associates each string to a vector of dimension $|\sum|^n$ containing the presence bits of all its sub-strings of length n. Ionescu and Popescu also extended this idea to *blended spectrum kernels*, which simply takes into account n-grams of different lengths and sums the corresponding kernels. So for instance, $k_{5-9}^{0/1}(s,t) = k_5^{0/1}(s,t) + k_6^{0/1}(s,t) \ldots + k_9^{0/1}(s,t)$. To ensure a fair comparison of strings of different lengths, a normalized version of the kernel was used, defined a follows:

$$\hat{k}_n^{0/1}(s,t) = \frac{k_n^{0/1}(s,t)}{k_n^{0/1}(s,s) + k_n^{0/1}(t,t)} \tag{3.2}$$

Note that $\hat{k}$ indicates that the normalized version of the kernel. This was the kernel used for the text based essays in the data set. The i-vectors were first normalized using the L_2 normalization, then the Euclidean distance was computed between each pair of i-vectors, and *then* the RBF kernel was employed to transform the distance into a similarity measure, which is defined as follows:

$$\hat{k}^{\text{i-vec}}(x,y) = exp(-\frac{\sqrt{\sum_{j=1}^{m}(x_j - y_j)^2}}{2\sigma^2}) \tag{3.3}$$

where σ was tuned in a set of preliminary experiments. Their final and best result was a combination of three kernels, namely $\hat{k}_{5-9}^{0/1} + \hat{k}_{5-7}^{0/1} + \hat{k}^{\text{i-vec}}$, where summing up kernels or kernel matrices is equivalent to feature vector concatenation [Ionescu and Popescu, 2017].

3.2.6 CEMI – Second place winners using stacked FFNNs

Finishing in the same top group as UnibucKernel was CEMI [Ircing et al., 2017] with a F_1 score of 0.926 in the fusion track. CEMI obtained their best results using a neural network based meta-classifier, meaning that they used several isolated feed-forward neural network models, each trained on a separate feature type. Features included word, character and POS n-grams, plus i-vectors. Finally the outputs were fused using softmax to predict the final label.

CEMI used TD-IDF weighting, capping the length of the feature vector by picking the top $30,000$ features entries sorted by descending TF-score. Ircing et al. report that using a size larger than $30,000$ did not increase performance. LDA was used for normalizing the i-vectors. Different ensemble architectures were tested, both homogeneous and heterogeneous. The homogeneous version worked best on the fusion track while a heterogeneous version worked well on the speech track. Very deep architectures were also employed, such as convolutional neural nets, DenseNets and ResNets. However, the simple FFNN was found to beat all of these models – possibly due to the size of the data set. Such deep architectures have a lot of weights that need to be learned, and need a lot of data in order to do so [Ircing et al., 2017].

Ircing et al. performed a thorough analysis of their results using local interpretable model-agnostic explanations (LIME). The analysis showed that typically certain content words can leak important information about the author. Typically, if the essay contained for example the words "Japan" or "Japanese", then the author was often from Japan, and similarly for other countries. The analysis also shows that spelling mistakes and typos have origin in the L1 language. For instance, it was common for Italians to write the word "public" as "pub**b**lic", from the Italian word "pubblico". Similarly for French authors, a common mistake was spelling "example" as "ex**e**mple", from the French word "exemple". As other shared task participants noted, close to 50% of the error confusions were between Hindi and Telugu. Ircing et al. assume that this was because the L1 speakers of these languages have gone through the same educational system in India.

3.2.7 ItaliaNLP – Winners of the essay track

The team called ItaliaNLP won the essay-only track of the 2017 with a F_1 score and accuracy of 0.882, which currently is the highest score achieved on the essay-only TOEFL11 test set. To obtain these results two stacked SVM-classifiers were used: One trained at the sentence level, and one trained on the document level, using the output of the sentence classifier as input, as well as the features from the documents themselves [Cimino and Dell'Orletta, 2017]. The features used were similar to the ones used by other systems: lexical features such as the text

length, average word length, character n-grams, function words, word n-grams and lemma n-grams. In addition both *morpho-syntactic* features such as POS n-grams, as well as syntactic features such as linear dependency n-grams were used.

3.2.8 NLI with User Generated Content

Quite recently, as in November 2018, Goldin et al. [2018] tried the task of NLI in a new environment using the large social media data set called the Reddit-L2 data set. In the experiments Goldin et al. used a simple regression classifier for all experiments. The main focus was not to build a new classifier for NLI, but to do feature exploration on the novel Reddit-L2 data set, as well as experiment with possible features specific to social media. The following subsection will cover the Reddit-L2 data set in more detail.

The Reddit-L2 Data Set

The Reddit-L2 data set[1] used by Goldin et al. [2018], is a novel data set originally created by Rabinovich et al. [2018]. The data is collected from a social media site called Reddit. Reddit is a platform where users can subscribe to areas of interest – so called "subreddits" [Singer et al., 2014]. The topics of subreddits range from very general topics such as "pictures"[2] and "science"[3] all the way to specific and obscure niche subreddits such as "birds with arms"[4], which is a subreddit created for posting pictures of birds that have been photoshopped to look like they have arms. In other words the texts produced by users of Reddit cover a very wide range of different topics. Within each subreddit, users can post both links and content such as videos and images, which other users can comment on. The site also features a voting system where other users can up- and down vote the content and comments of other users. Included in all these sub-communities there are some subreddits dedicated to Europe. Namely:

- `/r/europe`
- `/r/AskEurope`
- `/r/EuropeanCulture`

In these subreddits, the users can optionally provide a so-called *flair* which is a tag to indicate which country the user is from. This flair can be used to directly

[1] http://cl.haifa.ac.il/reports/L2/index.shtml
[2] www.reddit.com/r/pics/
[3] www.reddit.com/r/science/
[4] www.reddit.com/r/birdsWithArms/

infer the users L1. Rabinovich et al., the creators of the data set, performed experiments to assure the reliability of these labels, and Goldin et al. [2018] performed further measures to ensure that the labels are correct. In addition, Goldin et al. argued that incorrect labels will only function as noise to the model, making the task harder if so.

While previous data such as the TOEFL11 data set contain texts written by *learners* of English, many of the Reddit-L2 authors are close to fluent in English. This makes the task of performing NLI on this data set even more challenging. The data set consists of texts the users have produced in the Europe-related subreddits, as well as texts written in other subreddits. While the three European subreddits typically focus on topics related to Europe, additional subreddits can have virtually any topic. The data collected from the European-themed subreddits is deemed the *in-domain* data, while the data collected outside of these is called the *out-of-domain* data. The text data is separated into chunks, where each chunk contains 100 sentences from a single user. After pre-processing and removing users with less than 100 sentences (1 chunk), the data set consists of roughly $200,000,000$ sentences and 3 billion tokens across 29 native countries, from $34,511$ unique users.

Downsampling of the Reddit-L2 Data Set While the Reddit-L2 data set is large, the number of users varies a lot between countries, making the data set imbalanced with regards to classes. To combat this problem, Goldin et al. [2018] performed downsampling of the data. As the same downsampling will be used for the later experiments, the process will be explained in this section.

First and foremost, the downsampling starts by grouping the 29 native countries into 23 languages in total. This includes labeling users from the countries Ireland, UK, US, New Zealand and Australia into English, Austria and Germany into German, and Spain and Mexico are grouped as Spanish. After grouping, the full list of languages is English, German, Dutch, French, Polish, Romanian, Finish, Swedish, Spanish, Greek, Portuguese, Estonian, Czech, Italian, Russian, Turkish, Bulgarian, Croatian, Norwegian, Hungarian, Lithuanian, Slovenian and Serbian.

Furthermore, to achieve class balance, the number of users per label is capped at the number of users the label with the fewest users has. Lithuania and Slovenia have the fewest in-domain users (both 104), so for all other labels which have more users than this, 104 users are randomly selected for the label and the rest are discarded. This results in $104 \cdot 23 = 2392$ unique users, as there are 23 labels and 104 users per label. However, the number of chunks per user within each label is still uneven. To mitigate this, for each user, the median of the number of chunks for all users is randomly selected. For the in-domain scenario, the median is 3 chunks per user, and in the out-of-domain scenario the median is 17 chunks.

Feature Set	NLI Accuracy
Char. 3-grams	62.06
Token unigrams	31.26
Spelling	27.74
Grammar errors	8.36
Function Words	20.15
POS 3-grams	13.30
Sentence length	4.79
Social network	5.75
Subreddits	74.46

Table 3.2: The results obtained in Goldin et al. [2018] when running features individually on the Reddit-L2 corpus using logistic regression.

Evaluation

Goldin et al. [2018] used different evaluation strategies for the in-domain and the out-of-domain scenario. In both scenarios, the initial downsampling was executed, resulting in 2392 users total for each scenario. In the in-domain example, cross-validation was carried out by training on 90 % of the users in each fold, and testing on the remaining 10%, using only the in-domain data. Though it is not directly specified in the original paper, it is assumed that the reported accuracy is the average accuracy over these 10 folds.

For the out-of-domain scenario, 10% of the 2392 users were randomly selected uniformly across L1s. These 10% out-of-domain texts functioned as the test set. Then, the in-domain chunks of the remaining 90% of the users were used for training. This random sampling was carried out 10 times, and the average accuracy of the 10 runs was reported. Consequently, Goldin et al. [2018] never used the out-of-domain chunks for training, only for evaluation.

Features

As the main goal of Goldin et al. [2018] was to explore the novel data set, a description of the features used and results follows. A summary of all the features reported and their individual accuracies can be found in table 3.2.

Content features: Content features are features which capture information about what the author was writing about. For instance – as was observed during the 2017 shared task – authors of chunks that contain many words such as “Paris” or “Japan” tend to be from the corresponding countries (France and Japan,

respectively). The content features used in Goldin et al. [2018] were

- **Character 3-grams** (top 1000 most frequent)
- **POS 1-grams** (top 1000 most frequent)

Spelling and grammar features: Also included were features that could be inferred from an automatic spell checker, as it was hypothesized that native speakers tend to make less grammatical and spelling errors than non-natives. However, Goldin et al. do not mention the effect of web-browser spell checks, which could possibly help users spell words correctly which they might otherwise have misspelled. The spelling and grammar features included were:

- **Edit distance**. Average Levensthein distance between the original word and the correction offered by the spell checker, for all words in a chunk, as a feature.
- **Spelling errors**. Insertions, deletions and substitutions that yield the correct word from the misspelled word were extracted and used as features. This feature was limited to only the top 400-most frequent letter-to-letter substitutions.
- **Grammar errors**. This feature was expressed as a binary feature vector of length 2000, where each entry was one of the grammar rules obtained by the automatic spell checker.

Content-independent features: As opposed to the content-dependent features, some features are connected to how the author writes in general, regardless of the topic he or she is writing about. These features tend to capture more of the stylistic properties of the author, not what the text is about. The content-independent features used by Goldin et al. [2018] were:

- **Function words**. This feature was calculated by using the frequencies of the 400 most common function words.
- **POS 3-grams**. The normalized frequency of the top 300 most frequent POS 3-grams in the data set.
- **Sentence length**. The average length of the sentences in the chunk.

Social network features: In addition to the general NLI features listed above, Goldin et al. [2018] also experimented with social network specific features. As these features are not directly related to NLI in general, they will only be covered briefly. The features included the overall score users have based on up- and down votes from other users, the average score of the user per submission, the average number of submissions, the average number of comments, as well as the thirty most frequent subreddits the user had visited, compared to the most popular subreddits for each country.

Results

The performance of the logistic regression system was measured in accuracy, both in- and out-of-domain. Initially, each feature was considered individually, in domain, giving an intuition of their individual contribution to the overall results. The accuracy contribution of each individual feature set can be found in table 3.2. Some of the features alone performed barely over the random baseline, as the random baseline with 23 languages is $1/23 = 0.043$.

Furthermore the content features alone were tested in-domain and out-of-domain, and the accuracy dropped to nearly half when being tested outside the European domain. The content-independent features performed worse in general, but the drop in accuracy was not as substantial when tested out-of-domain. These results were expected, as the content dependent features reap benefits mostly when inside the domain, but lose their ability to discriminate once taken outside the domain. The grammar and spelling errors performed similarly to the content-independent features.

The most visited subreddits feature performed stunningly, both in and out of the domain. However, as the authors acknowledge, this feature is not only specific to the particular data set, but as many users tended to frequent subreddits specific to their country (someone from Norway might be subscribed to `/r/Norway` for example) this feature often contained the correct label itself. This allowed the model to peek at the correct labels in many cases, which explained why the Subreddit-feature alone could achieve such high accuracies.

Acknowledging this caveat, Goldin et al. reported the results of using all features, excluding the Subreddit feature. Removing the Subreddit feature yielded a final accuracy of 0.690 in the in-domain scenario, dropping down to 0.36 when tested out-of-domain.

3.2.9 NLI with Classifier Stacking and Ensembles

Malmasi and Dras [2018] performed a systematic examination of ensemble methods for NLI, in addition to evaluating deeper ensemble architectures such as classifier stacks. Malmasi and Dras [2018] presented a set of experiments using

three ensemble-based models, testing each with multiple configurations and algorithms. The experiments included a rigorous application of meta-classification models for NLI, and achieved state-of-the-art results on several large data sets, evaluated both intra-corpus and cross-corpus.

In the initial experiments, Malmasi and Dras [2018] experimented with single SVMs and logistic regression models on the TOEFL11 test set, for different feature types. The feature types used were character 1-3 grams, function word 1-2 grams, lemmas 1-2 grams and word 1-2 grams. Additional features such as POS tags and context-free grammar rules were also used, but did not perform too well compared to the other features. The initial experiments showed that the SVM model performed better than the logistic regression model, and SVMs were consequently used as the base classifier for the rest of the experiments.

Next, the base SVMs were used in a voting ensemble. The mean probability voting rule performed best, closely followed by the plurality vote and median probability vote.

Next, a selection of meta-classifiers were applied to both the discrete one-hot-encoded outputs of the base models, as well as the continuous probability outputs. Two important trends were observed based on the meta-classifier experiments: The meta-classification results were better than the ensemble combination methods alone, and the meta-classifiers trained on continuous output performed better than the discrete label counterparts. Malmasi and Dras [2018] assume the latter was because the continuous outputs of the base models provided the meta-classifier with more information than using discrete outputs. SVMs linear regression meta-classifiers performed well, while using a FFNN meta-classifier performed slightly below the other models. However, the best performing meta-classifier was LDA, which obtained an accuracy of 0.868 on the TOEFL11 test set.

Furthermore, experiments using ensembles of meta-classifiers were also carried out. The experiments showed that bagging ensembles performed best empirically, and again that an ensemble of meta LDA classifiers obtained the best results. In fact, the ensemble of LDA meta-classifiers obtained an accuracy of 0.871 – close to the state-of-the-art accuracy of 0.882 obtained by ItaliaNLP on the same test set during the 2017 shared task [Cimino and Dell'Orletta, 2017].

Malmasi and Dras performed additional experiments on other copra of both English and other language, both intra-corpus and cross-corpus, and achieve state-of-the-art results on these as well.

3.3 BERT

Devlin et al. [2018] introduce a new attention-based language representation model called BERT, which stands for Bidirectional Encoder Representations

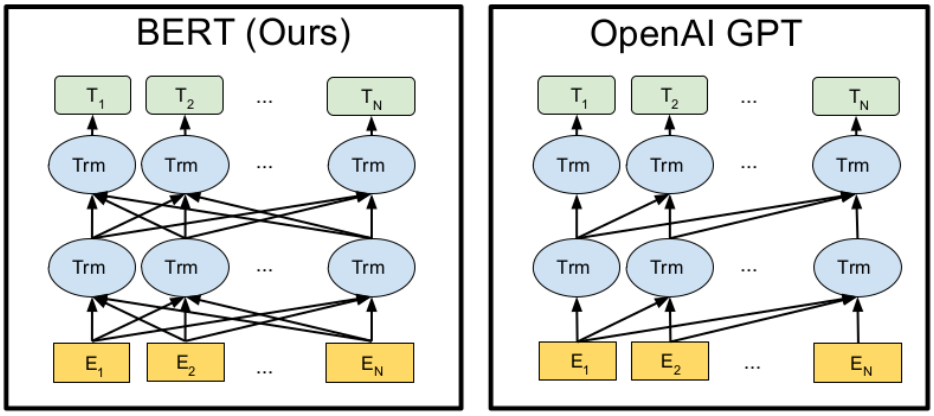

Figure 3.1: A comparison of a left-to-right model on the right, compared to the bidirectional BERT architecture on the left. Figure from Devlin et al. [2018], with permission from Jacob Devlin.

from Transformers. BERT uses multiple layers of Transformer-models to encode sequences into meaningful representations. These representations can be pre-trained on general tasks, and can later be fine-tuned with just one additional output layer. This additional layer can be used to create state-of-the-art models for a wide range of tasks, without substantial task-specific architecture modifications. Devlin et al. show that BERT is empirically powerful on tasks ranging from question answering to natural language inference.

Similar fine-tuning approaches have been made previously. However, a major limitation for these approaches has been that standard language models are unidirectional. This limits the choice of architecture that can be used during pre-training, and can severely restrict the power of the pre-trained representations [Devlin et al., 2018]. This because every token can only attend to previous tokens in the self-attention layer of the Transformer. In order to address the issue of bidirectionality, Devlin et al. introduce a new, multi-layered bidirectional architecture, as well as two unsupervised tasks which require the model to attend to the entire sequence, not just predecessing tokens. These two pre-training tasks will be described in more detail in section 3.3.2. A visualization of how BERT's bidirectionality differs from previous systems can be seen in figure 3.1.

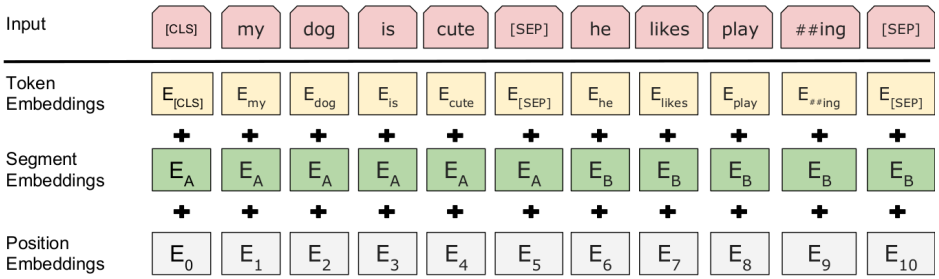

Figure 3.2: An overview of how BERT's input embeddings are composed. The final input sequence is a result of the concatenation of position embeddings, A and B segment embeddings, and WordPiece token embeddings. All three types of embeddings are learned during training. Figure from Devlin et al. [2018], with permission from Jacob Devlin.

3.3.1 BERT Model Architecture

BERT mainly consists of multiple layers of fully connected Transformers. The general architecture is illustrated on the left in figure 3.1. Devlin et al. [2018] provide and experiment with two versions of the BERT model. The first one is called BERT-base, and consist of 12 layers (Transformer blocks), where each Tranformer has 12 attention heads, as opposed to the 8 attention heads used in Vaswani et al. [2017]. The hidden size is set to 768, which yields a total of 110 million parameters. The next model, named BERT-large, consists of 24 Tranformer layers, a hidden size of 1024 and 16 attention heads per Transformer, resulting in a total of 340 million parameters.

BERT uses WordPiece token embeddings as input, with a vocabulary of $30,000$. In addition to just learning the token embeddings, BERT also learns two segment embeddings, E_A and E_B, as well as position embeddings. The final input is created by concatenating the three embedding types into a single input sequence, as shown in figure 3.2. The two segment embeddings, E_A and E_B, allow BERT to handle both single sentence input, as well as sentence pairs. This is done by separating sentence pairs by a special separator token (`[SEP]`) and adding embedding E_A to the embeddings which belong to the first sentence, and E_B to the tokens belonging to the second sentence. For single sentence inputs, the E_B embedding is not used, and only the E_A embedding is added. For the pre-trained models, positional embeddings are learned for sequence lengths of up to 512 tokens.

The first token of every sequence is always the special classification embedding, `[CLS]`. For text classification, the final hidden state is the output of the

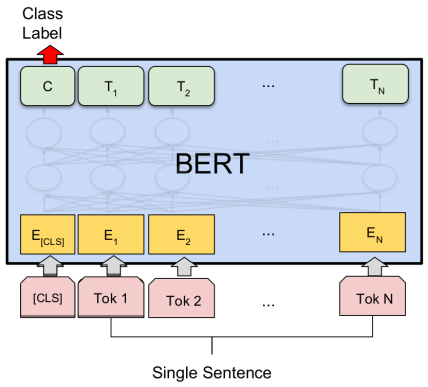

Figure 3.3: An illustration of how BERT is used for text classification. A single sentence (a sequence of tokens) is fed as input and processed into embeddings. These embeddings are fed through the network of transformers, and the final hidden state of the first transformer is used for text classification. Figure from Devlin et al. [2018], with permission from Jacob Devlin.

Tranformer corresponding to the `[CLS]` token. This process is illustrated in figure 3.3. This final output can then be used for other classifiers for text classification or other purposes.

3.3.2 BERT Pre-Training Tasks

As mentioned initially, BERT is pre-trained on two different tasks, both of which require the model to attend to both the right and left context of the target. In the first task, the language model masks some of the tokens of the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to-right language model pre-training, this objective allows the resulting presentation to fuse the left and the right context.

In practice, 15% of all WordPiece tokens are masked at random, and the model is trained to predict the masked words based on the entire sentence. A problem with this approach is that there is a mismatch between the pre-training and fine-tuning, since the `[MASK]` token never appears during fine-tuning. In order to mitigate this effect, the masked word is not always replaced by the masked token. Instead, some minor alterations are made. More specifically, the language model performs the masking in the following way:

- Pick 15% of the tokens at random.

- 80% of the time, replace the word with the `[MASK]` token.
- 10% of the time, replace the word with a random word in the vocabulary.
- 10% of the time, keep the word unchanged.

The purpose of the final 10% is to bias the representation towards the actual observed word. Devlin et al. [2018] report that the random replacement does not seem to hurt the language understanding capability of the model, as the replacement only occurs for 1.5% of all tokens, (10% of 15%).

The second task is called next sentence prediction, a binary task which jointly pre-trains text-pair representations. The task is simple, yet effective: given two sentences A and B in the corpus, predict whether sentence B is the succession sentence of sentence A. During pre-training, the model is trained on sentence pairs where 50% of the time sentence B is the actual sentence following A, and 50% of the time it is not. Devlin et al. [2018] report between 0.97 - 0.98 accuracy on this task after pre-training.

The pre-training of BERT was carried out using the two abovementioned tasks on the concatenation of two corpra. The first is a corpus called the BooksCorpus (800 million words), and the second one is the English Wikipedia (2,500 million words). The pre-training was carried out on 4 cloud TPUs (Tensor Processing Units) in Pod configuration, resulting in 16 TPU chips total. Under these settings, the training of BERT-base and BERT-large took 4 days each, and the final pre-trained models were made publicly available [5] for anyone to use.

3.4 Motivation

Now that the best performing systems of NLI and the details of the attention-based BERT architecture have been covered, a brief section has been dedicated to sum up the findings of the literature review and to motivate the experiments of the next chapters. Some important observations made in the literature review were:

1. The best performing systems for NLI to date all use some kind or ensemble or classifier-stacking approach [Malmasi and Dras, 2018; Malmasi et al., 2017].
2. Some of the most common features for NLI are character-, word- and lemma n-grams [Malmasi et al., 2017]. Additionally, content dependent features, such as function words, seem to suffer less when being tested in a different domain than trained in [Goldin et al., 2018].

[5] https://github.com/google-research/bert

3. Deep learning has not yet proven to be able to beat traditional classifiers on the task of NLI. It is hypothesized that this is due to the lack of data available for the task [Ircing et al., 2017].

4. The recent release of the Reddit-L2 data set provides a lot more data for the task of NLI. The data includes texts of more proficient writers of English [Goldin et al., 2018].

5. The novel, attention-based, deep learning architecture named BERT has empirically been shown to provide state-of-the-art results on several NLP tasks Devlin et al. [2018].

These observations bring up some interesting questions. First of all, if BERT provides state-of-the-art results on several NLP tasks, can it do the same on the task of NLI? One problem with applying BERT to NLI is that the lack of data could be a challenge, as with previous deep learning approaches. However, with the much larger Reddit-L2 data set now available, the problem of lacking data can be mitigated. Furthermore, as ensemble methods and meta-classifiers have proven to increase performance on the task of NLI, can ensembles combining both the traditional state-of-the-art techniques as well as BERT yield even better performance?

All of these concerns are highly relevant for reaching the main Goal and answering the Research Questions of this Thesis. The following chapter will propose a BERT-model suited for the task of NLI, as well as two meta-classifier architectures combining BERT and traditional classifiers.

Chapter 4

Architecture and Model

The following chapter describes three model architectures which will be used in the experiments in chapter 5. First, a stand-alone BERT architecture suited for the task of NLI will be covered in section 4.1. This model will be used to answer Research Question 1 and 2, which are concerned with how attention-based systems perform on the task of NLI in isolation. Next, section 4.2 proposes two architectures, namely a meta-classifier architecture and an ensemble of meta-classifiers. These stacked classification architectures use traditional techniques in combination with BERT in order to answer Research Question 3, regarding how attention-based systems can be used in combination with existing techniques in order to increase performance on the task of NLI.

4.1 BERT Model Architecture

The overall BERT architecture consists of the pre-trained BERT-base model provided by Devlin et al. [2018], as well as a linear layer put on top of the model, as suggested by Devlin et al.. While the BERT model is pre-trained, the linear layer must be trained from scratch for each classification task. The linear layer is randomly initialized at the start of training.

For each example instance, the sequence is fed through BERT, and the produced output is fed through the external linear layer. Thus the linear layer has an input size the same as the BERT hidden size output. The output size of the linear layer is set to be the same as the number of labels for the relevant task (11 for TOEFL11 and 23 for Reddit-L2). The BERT-base model and the linear layer are then trained trained together using the cross-entropy loss with regards to the correct label.

4.1.1 BERT Casing

In all experiments, the uncased version of BERT has been used, meaning that the BERT model has only been trained on lowercase text. The reason the cased version of BERT has not been used is to reduce the complexity of pre-processing tokenization, as well as reducing the number of experiments required.

It is hypothesized that using the cased version of BERT for NLI might be beneficial in some cases, but at the expense of other practicalities. For instance, in English, the names of the months are always capitalized (January, February and so on), while in Norwegian they are not. Cases like this could perhaps help the model when discriminating L1s to some extent. For instance, an author with Norwegian as their L1 might forget to capitalize the name of the months when writing English, which is information the model could utilize.

However, retaining capitalization would also increase the vocabulary size, as all words containing capitalized words would require an embedding of their own. Alternatively, keeping the vocabulary size fixed would simply reduce the number of unique words available, as one word would have to be represented both in lower- and upper case. For the above-mentioned reasons, all experiments are limited to the uncased version of BERT.

4.1.2 BERT Hyper-Parameters

When obtaining state-of-the-art results on several NLP tasks, Devlin et al. use the same hyper-parameters for all experiments, except for three: The batch size, the Adam learning rate, and the number of epochs. For these remaining values, Devlin et al. provide the following possible ranges which they found to work well across different tasks:

- **Batch Size** 16, 32
- **Learning Rate (Adam)** 5e-5, 3e-5, 2e-5
- **Number of epochs** 3, 4

The experiments using the BERT model will follow these recommendations, only varying the three hyper-parameters listed above. The rest of the hyper-parameters will remain fixed. A full list of the hyper parameters of the BERT-model can be found in table 4.1, with parentheses indicating ranges of possible values. The only hyper-parameter which will not follow the recommended value is the maximum sequence length, which has been set to the largest possible value, 512 – the reason for which will be described in more detail in the following section.

Hyper Parameter	Value	Description
Hidden Layers	12	
Hidden Size	768	
Heads	12	Number of attention heads per transformer.
Epochs	(3, 4)	
Learning rate	(5e-5, 3e-5, 2e-5)	The initial learning rate for Adam.
Batch size	16	
Maximum sequence length	512	The maximum total input sequence length after WordPiece tokenization.
Warmup proportion	0.1	Proportion of training to perform linear learning rate warmup for.
Gradient accumulation steps	1	Number of steps to accumulate before performing a backward/update pass.

Table 4.1: The hyper-parameters used for BERT in all experiments, unless stated otherwise. Parentheses indicate a range of possible values.

4.1.3 Deciding BERT's Maximum Sequence Length

One of BERT's hyper-parameters is the maximum input sequence length, for which the recommended value is 128. However, for a text document classification class such as NLI, this is not ideal. Most documents available in the NLI-data sets contain far more tokens than 128, which means that a lot of information in each document will be thrown away if this value is used. In order to get a notion of how long each document in the available data is, the average and maximum number of WordPiece tokens per document has been counted and calculated in table 4.2. For the TOEFL11 data set, almost 99% of all documents contain more than 128 tokens. However, less than 5% of the documents have more than 512 tokens. The 5% of documents which have a sequence length longer than 512 will lose some information, but the effect on performance should be negligible. For this reason, the maximum sequence length for BERT has been set to 512 for all experiments, in order for BERT to take full advantage of the documents in their entirety. 512 is the maximum possible value for the sequence length, as the pre-trained model has not been trained for longer sequences. For the Reddit-L2 data set, however, each document far surpasses 512 tokens. For this reason, the documents will have to be split up into texts of roughly 512 tokens in order to not waste data.

Data Set	Max. Tokens	Avg. Tokens	>128	>512
TOEFL11	910	369	98.94%	4.9%
RedditL2	18149	2072	100%	100%

Table 4.2: Table showing the maximum and average number of WordPiece tokens per document in the TOEFL11 training set and Reddit-L2 in-domain data set. Columns 4 and 5 show the percentage of documents which have a sequence length greater than 128 and 512, respectively.

4.1.4 Memory Issues and Batch Size

When implementing the BERT model, some preliminary experiments were run to check whether the code compiled. During these preliminary experiments it became clear that BERT-large with a maximum sequence length of 512 caused memory issues with the GPUs available (see section 5.2.4 for more information on the resources available). Changing the batch size to 32 for BERT-base also caused the same memory errors. With a sequence length of 512, the largest models able to run with the resources available were BERT-base with a batch size of 16 or less, and BERT-large with a batch size of 1. Any batch size larger than 1 with a sequence length 512 caused the aforementioned memory issues.

For this reason, all experiments will only use the BERT-base model, except for one: running BERT-large with a batch size of 1, mostly as a curiosity. This is unfortunate, as Devlin et al. [2018] report that BERT-large provides a significant boost in performance over BERT-base, even for small data sets. Additionally, as BERT-base with a batch size of 32 caused the same issues, only batch sizes of 16 or smaller were considered.

4.2 Meta-Classifier Architectures

As the main goal of this Master's Thesis is to explore how attention-based systems can be used to improve performance on the task of NLI, and Research Question 3 is specifically concerned with how an attention-based system can be used in union with the current-state-of-the art approaches, two novel architectures are proposed. The architectures are both heavily inspired by the classifier-stacking of Malmasi and Dras [2018], but include BERT as an optional base-classifier. The first architecture is a stacked-architecture consisting of homogeneous base classifiers, which combined output is fed into a meta-classifier, which provides the final decision of the stack. The second architecture is similar to the first, but instead of using a single meta-classifier, an ensemble of meta-classifiers is applied to provide the final decision of the stack.

4.2.1 Meta-Classifier Architecture

An illustration of the proposed meta-classifier architecture can be found in figure 4.1. The inclusion of BERT as a base-classifier is optional, indicated by stippled lines. The stack starts at the left of the illustration, where all T base classifiers are fed the raw text input. The base classifiers are all homogeneous, except for BERT. Each base classifier is trained on different feature types, which is indicated by the preliminary F_i boxes, which transform the raw text into different types of feature-vectors. After training, each base classifier produces a vector of probabilities for each class. Malmasi and Dras [2018] experiment with using both the continuous probability output of the models as well as the discrete one-hot encoded representation of the labels, and find that outputting continuous probabilities yields better performance in general. Based on this observation, the meta-classifier architecture will do the same. Next, all the probaility vectors are concatenated, creating a 1 x $T \cdot num\_classes$ or a 1 x $(T+1) \cdot num\_classes$ vector, depending on whether BERT is included or not. Finally, the meta-classifier is trained on the concatenated output of the base classifiers and the original training labels, to produce a single prediction per example instance.

4.2.2 Ensemble of Meta-Classifiers Architecture

An illustration of the proposed stack with an ensemble of meta-classifiers can be found in figure 4.2. Most of the architecture's details are the same as the single meta-classifier architecture described in the previous subsection. The key difference is that the single meta-classifier is replaced with an ensemble of meta-classifiers. These meta-classifiers are put together in a bagging-ensemble, where all models are trained on different subsets of the outputs of the base classifiers. The bagging ensemble has been chosen as it was the best performing ensemble found in Malmasi and Dras [2018].

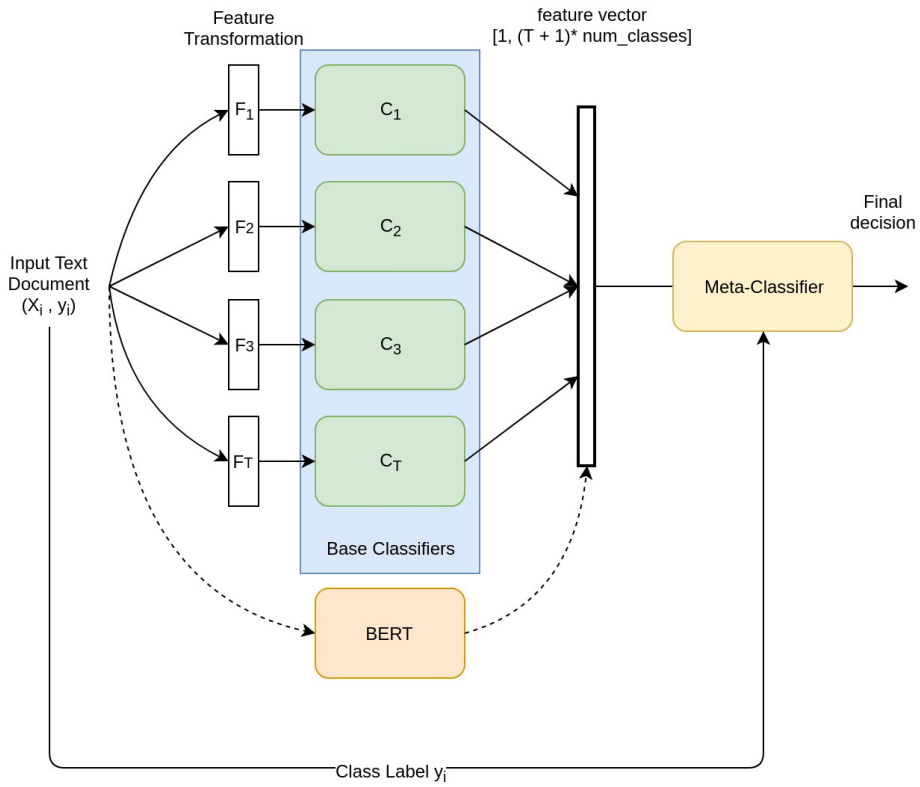

Figure 4.1: An illustration of a meta-classifier stack. All base classifiers, including BERT, are fed the same input. The concatenated output is fed into the meta-classifier which produces the stack's final decision.

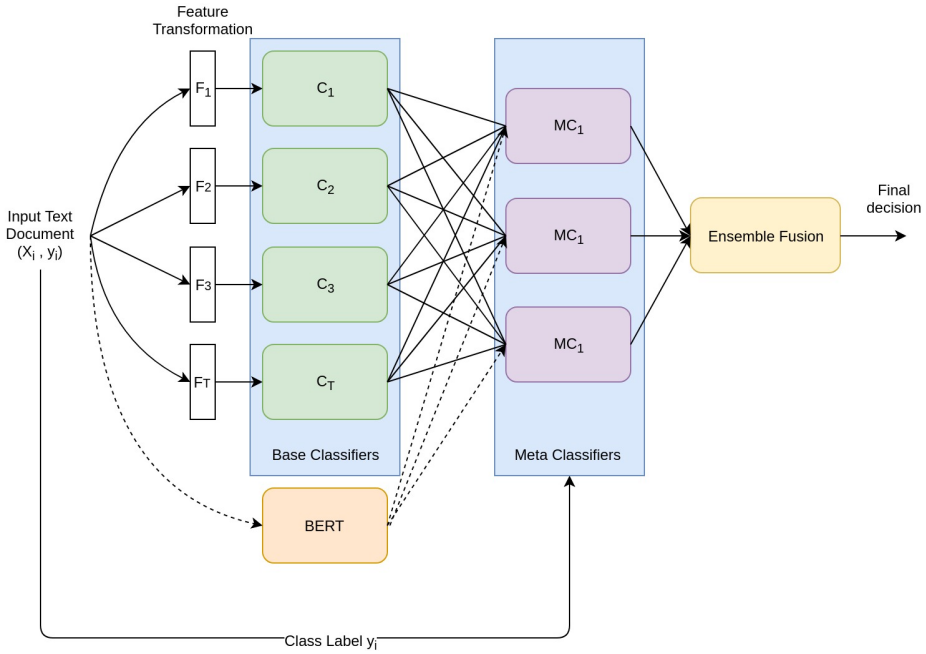

Figure 4.2: An illustration of a stack with an ensemble of meta-classifiers. All base classifiers, including BERT, are fed the same input. The concatenated output is fed into the ensemble of meta-classifier which produces the stack's final decision.

Chapter 5

Experiments and Results

The following chapter will present all experiments, and their corresponding results. First, an experimental plan is proposed. In this section each experiment, as well as what Research Question the experiment is designed to answer, is explained in detail. Next follows a section covering the details of the experimental setup, including technologies used and the environment the experiments were run in. Finally, the results of each experiment are presented and briefly discussed.

5.1 Experimental Plan

5.1.1 Baseline Experiments

The first experiment will simply work as a baseline for comparison of future results. For this initial experiment, a simple multinomial Naive-Bayes and an SVM classifier will be trained and evaluated on the unigram representation of the documents. As in Malmasi and Dras [2018] and other work, the unigram feature vector will be normalized using the TF-IDF score of each word. The Baseline Experiments will be carried out on both TOEFL11 and the Reddit-L2 data sets. The exact details of the data used will be covered in the following experiments. Naive-Bayes will be used as a baseline, as it is a common baseline classifier for text-classification tasks. Additionally, an SVM will serve as a more demanding baseline – as the related work section has shown that simple SVM architectures already work quite well on the task of NLI. Word unigrams have been chosen since it is a simple feature type, well suited for baseline experiments. Additionally, the random baselines for each test scenario will be provided. The random baselines are the probabilities of guessing the correct prediction by choosing any of the given classes at random.

5.1.2 Experiment 1 – Initial Experiments

Experiment 1 is designed to directly answer Research Question 1 – how an attention-based system alone performs on the task of NLI. For these experiments, BERT will be trained and tested, both on the TOEFL11 data set and the Reddit-L2 in-domain scenario. Additionally, these initial experiments will serve as an indicator of what hyper-parameters the model performs best under.

Experiment 1a

As most of the related literature trains on the TOEFL11 training set and evaluates on the official TOEFL11 test set, an initial experiment following the same setup using BERT will be carried out. This initial experiment will give an indication of how an attention-based architecture performs on the task of NLI in isolation. The results obtained in Experiment 1a should be directly comparable to the results reported by most of today's systems. In this initial experiment, the BERT model described in section 4.1 will be used, and all hyper parameters will be set to the default values listed in table 4.1 from section 4.1.2. However, the hyper-parameters with several recommended values (sequence length, learning-rate and the number of epochs) will be varied, one parameter at a time. As discussed in section 4.1.4, running any BERT model with a batch size larger than 16 caused memory issues with the devices available. Because of this, the initial batch size will be kept constant at the recommended value of 16, and only the number of epochs and the learning-rate will be varied. Additionally, a learning rate of 4e-5 will be added for completion, and 5 epoch experiments will also be run in order to see to what extent the model is overfitting as the number of epochs increases. With 3 possible epoch values and 4 different learning rates to choose from, a total of $3 \cdot 4 = 12$ different experiments must be run. After these initial experiments have been carried out and the best performing settings have been found, experiments using batch sizes below 16 will be run in order to explore the impact of the batch size. Additionally, mostly as a curiosity, the BERT-large model will be run with a batch size of 1, under the optimal parameter settings found for BERT-base, to see if a larger model yields better results.

Experiment 1b

Similarly to Experiment 1a, the same experiment will be carried out on the Reddit-L2 in-domain data set. In order to make the results comparable to other work carried out on the same data set, the same downsampling and 10-fold cross-validation as the in-domain scenario described in Goldin et al. [2018] will be performed. Due to the size of the data set and the fact that 10-fold cross-validation requires training a model 10 times, only one experiment will be carried out for the

Sub-Chunk	Sub-Chunk Pred.	Chunk Pred.	Correct
`username_chunk1_0`	English	English	English
`username_chunk1_1`	English		English
`username_chunk1_2`	German		English

Table 5.1: Table showing an example of how the heuristically split sub-chunks are recombined into a single prediction for the original chunk as a whole by majority vote. In this example, the sub-chunk accuracy would be $2/3 = 0.667$, while the accuracy evaluated on the original chunk would be $1/1 = 1.0$.

Reddit-L2 data set, as opposed to running the 12 experiments from Experiment 1a twice with new data. Instead, the hyper-parameter settings which performed best on the TOEFL11 data set in Experiment 1a will also be used for the Reddit-L2 experiments.

Division of Sub-Chunks for Reddit-L2 As discussed in section 4.1.4, all documents in the Reddit-L2 data set have far more tokens than BERT's maximum sequence length of 512. For this reason, a heuristic division of the documents will be applied. The heuristic will simply be to divide all examples into smaller sub-examples (or sub-chunks) with a length of roughly 512 tokens by splitting on spaces. For instance, if a user with the username `username1` has a `chunk_1` with 1500 tokens in total, this chunk would be divided into three sub-chunks of roughly 512 tokens each, namely: `username1_chunk1_0`, `username1_chunk1_1` and `username1_chunk1_2`. After training has been performed on the sub-chunks, evaluation will be carried out both on the individual sub-chunks, and on the recombination back into the original chunks. The latter will be done by using a majority vote based on each sub-chunk prediction, for the entire chunk. An example of this recombination can be found in table 5.1. In this example, the model predicts two parts of the original chunk to be English, and the final part of the chunk to be German. This would give a sub-chunk accuracy of $2/3 = 0.667$. However, as English is the most predicted label for all the sub-chunks, the majority prediction for the chunk as a whole would be English, which gives a correct prediction for the document and a chunk accuracy of $1/1 = 1.0$. Ties are broken randomly. It is important to notice that the previous work on the Reddit-L2 data set done by Goldin et al. [2018] evaluates on the prediction of each chunk. For this reason, the accuracies obtained at the chunk level will be the comparable measurement of the systems performance, not the ad hoc sub-chunks, which have only been created in order to fit the limitations of BERT.

5.1.3 Experiment 2 – Testing Out-of-Domain Robustness

Experiment 2 is designed to directly answer Research Question 2 – how an attention-based system performs on the task of NLI when tested on documents concerning topics different from what it was initially trained on. In order to explore this, the out-of-domain experimental setup described in Goldin et al. [2018] and explained in section 3.2.8, will be followed, using the same BERT-architecture as in Experiment 1a and 1b. The model will be trained on the in-domain data, and tested on out-of-domain data of different users. Again the model will be trained under the optimal hyper-parameter settings found in Experiment 1a. This way, the results will be directly comparable to the in-domain results obtained in Experiment 1b, which will show how much the model suffers when trained out-of-domain.

5.1.4 Experiment 3 – Combining Attention With State-of-the-Art Techniques

Experiment 3 is designed to answer Research Question 3 – whether the combination of an attention-based system and the techniques used in the current state-of-the-art can improve performance on the task of NLI. Experiment 3 will use the meta-classifier and ensemble of meta-classifiers architectures described in section section 4.2.1 and 4.2.2, respectively. A grid search over several hyper-parameters will be carried out, namely the different base classifiers, different types of features per base classifier, and the maximum number of features per base classifier. First, the different models and feature types will be explored individually, in order to assess their individual contribution to the meta-classifier or ensemble. Both SVMs and FFNNs will be used as base classifiers. SVMs, as the literature review showed, are still the most popular and best performing classifiers for NLI. FFNNs have been included as they are typically quick to train, and showed promising results in Ircing et al. [2017]. As for features, word 1-3 grams, character 1-4 grams and lemma 1-2 grams will be used, in addition to the content-independent function word 1-2 grams. The features chosen are based on the features used in Ircing et al. [2017] and Malmasi and Dras [2018]. Similarly to Malmasi and Dras [2018], the main focus will not be on feature exploration, but instead using a predefined set of features, and evaluating whether the inclusion of BERT can improve the results obtained by these features alone.

After the individual base classifiers have been assessed, the continuous outputs of the models will be used as input for the single meta-classifier, as well as for the ensemble of meta-classifiers. In these experiments, both SVMs and FFNNs will be tested as meta-classifiers. All experiments will be run on both the TOEFL11 data set and the Reddit-L2 data set – both in- and out-of-domain. The meta-ensemble will have the same structure as the best performing ensemble

described in Malmasi and Dras [2018]. That is, a bagging ensemble using 200 meta classifiers, each trained on 80% of the total training data. All experiments will be run both with and without the inclusion of BERT as one of the base classifiers. The latter will allow for evaluation of whether the attention-based classifier can help improve performance on the task, as opposed to only using traditional techniques.

5.1.5 Experiment 4 – More Data

While the Reddit-L2 data set is huge compared to previous data sets used for NLI, most of the data is discarded when performing downsampling. The downsampling is done in order to maintain class balance. However, using evaluation metrics which take class imbalance into consideration – such as the macro-averaged F_1 score – the problem of class imbalance can be mitigated. As discussed by Ircing et al. [2017] and others, lack of data is hypothesized to be one of the main reasons traditional models still perform better than deep learning models.

In light of this, experiment 4 will be concerned with utilizing as much data as possible. Experiment 4 will be to train BERT on the out-of-domain Reddit-L2 data, and then use the entire in-domain data set for testing. This experiment will show how the same attention-system as used in Experiments 1a, 1b and 2 performs with more data available. Additionally, training on only the out-of-domain data will display how the system reacts when trained on a wide range of topics, and tested on different ones.

The out-of-domain training data will be engineered to be roughly 10 times the size of the in-domain test set in order to maintain a fair training-to-test ratio, and will be balanced to contain roughly the same number of examples per label. In practice, this is done by capping the maximum number of training sub-chunk examples per label to be $80,000$. This heuristic division will cause some class-imbalance in the training-data, as some languages have less than $80,000$ sub-chunks. The most extreme case is Slovenian, which has $24,787$ sub-chunks available out-of-domain. This issue will have to be tolerated, however, as the intent of Experiment 4 is to explore how more data can impact the performance of the system. An alternative solution would be to set the cap at $24,787$ sub-chunks per class. This would result in $24,787 \cdot 23 = 570,101$ training sub-chunks, as opposed to close to three times as many when setting the cap at $80,000$. Setting the cap at $80,000$ might bias the model towards the largest classes such as English, Dutch and German. However, as most classes contain more than $80,000$ sub-chunks, the problem should not be too significant. Additionally, using macro-averaged F_1 as the evaluation metric will measure the model's performance with the same weighting for each class, accounting for such a bias to some extent.

5.2 Experimental Setup

In order to make the results provided reproducible, the following section is dedicated to the details of the experimental setup used, in addition to libraries and tools used for the implementation of all experiments. The code for the entire Master's Thesis is readily available at GitHub[1].

5.2.1 BERT Implementation

BERT was implemented using PyTorch[2], an open source deep learning platform, and the PyTorch implementation of BERT[3]. The latter is a PyTorch clone of the official BERT implementation[4] released by Google for a different deep learning platform called TensorFlow. The PyTorch implementation has empirically been shown to give comparable results as the original implementation, and uses the same pre-trained BERT models as provided by Devlin et al. [2018].

5.2.2 SVM, FFNN and Other Tools

All traditional classifiers were implemented using the open source Scikit-Learn[5] (Sklearn) Machine Learning library. In addition, Sklearn was used for feature manipulation, such as tranforming text into TF-IDF weighted vectors, etc. For SVMs the default parameters of Sklearn[6] were used, but with a linear kernel, as it was the kernel used for all base classifiers in Malmasi and Dras [2018]. All FFNNs were run with the default parameters of Sklearn[7], unless specified otherwise. For manipulation of data and keeping track of the outputs of the base classifiers, the open source Pandas Data Analysis library[8] was used. For basic text-processing and feature manipulation (such as creating function and lemma words) the open source Natural Language Toolkit (NLTK[9]) was used.

5.2.3 Data Sets

For all experiments, either the TOEFL11[10] or the Reddit-L2[11] data set have been used, both of which are openly available. Regarding the TOEFL11 data set, the

[1] https://github.com/stianste/BERT-NLI
[2] https://pytorch.org/
[3] https://github.com/huggingface/pytorch-pretrained-BERT
[4] https://github.com/google-research/bert
[5] https://scikit-learn.org/stable/
[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
[8] https://pandas.pydata.org/
[9] https://www.nltk.org/
[10] https://catalog.ldc.upenn.edu/LDC2014T06
[11] http://cl.haifa.ac.il/projects/L2/index.shtml

raw text documents have been used. As for the Reddit-L2, the Reddit-L2 chunks were used, as in Goldin et al. [2018].

5.2.4 Environment and Resources

All experiments were carried out on the NTNU HPC Idun Cluster[12]. Running jobs on Idun allows for the use of two GPUs, both with 16 GB each, which were used for all experiments which required training of BERT. For most jobs, 64 GB RAM was sufficient, but for the largest job (Experiment 4) 128 GB RAM was needed in order to keep both BERT and all the data in memory. Running BERT on TOEFL11 for 5 epochs typically took 2-3 hours, while the final experiment using millions of Reddit-L2 examples took more than 6 days to run.

5.3 Experimental Results

5.3.1 Results of Baseline Experiments

Classifier / Data Set	Naive-Bayes	SVM	Random Baseline
TOEFL11 Test	0.559	0.726	0.091
Reddit-L2	0.377	0.716	0.043
Reddit-L2†	0.350	0.574	0.043
Reddit-L2*	0.176	0.400	0.043
Reddit-L2*†	0.169	0.322	0.043

Table 5.2: Table showing the accuracies obtained by the baseline classifiers when trained using TF-IDF weighted unigrams on the TOEFL11 and Reddit-L2 data sets. Asterix indicates the out-of-domain scenario of Reddit-L2, and † indicates that the model was evaluated on sub-chunks. For TOEFL11, both classifiers were trained on the training set and evaluated on the TOEFL11 test set. For the Reddit-L2 experiments, the average accuracy over a 10-fold cross-validation is reported.

The results of the Baseline Experiments can be found in table 5.2. Both classifiers clearly perform better than the respective random baslines, and the SVM clearly outperforms the Naive-Bayes classifier, as expected based on related work. The simple unigram SVM baseline outperforms the best accuracy of 0.690 obtained on the Reddit-L2 in-domain scenario using logistic regression in Goldin et al. [2018]. The unigram SVM also beats the previous best out-of-domain score of

[12]https://www.hpc.ntnu.no/display/hpc/Idun+Cluster

0.362 with an accuracy of 0.4. The accuracy of both models drop when tested in the out-of-domain Reddit-L2 scenario, and further drops when the models are evaluated on the more granular sub-chunks.

5.3.2 Results of Experiment 1a

Learning-rate / Number of epochs	**3**	**4**	**5**
2e-5	0.749	0.760	0.759
3e-5	0.746	0.765	**0.777**
4e-5	0.736	0.760	0.761
5e-5	0.734	0.746	0.765

Table 5.3: Table showing the accuracies obtained by BERT for different number of epochs and learning rates, trained on the TOEFL11 training set and evaluated on the TOEFL11 test set.

The results of training and testing BERT on TOEFL11 can be found in table 5.3. The results are also visualized in figure 5.1. As can be seen in the figure, increasing the number of epochs typically increases the final accuracy, regardless of the learning-rate used. The best results are obtained using a learning-rate of 3e-5 for 5 epochs, with a final accuracy score of **0.777**. Under these hyper-parameters, the model obtained a final training loss of 0.110, and an evaluation loss of 0.824. Out of curiosity, the same experiment was also run with the same parameters for 10 epochs, in order to see how the model performs after an even higher number of epochs. The 10 epoch model achieved an accuracy slightly below the best one at 5 epochs, with a training loss of 0.004. However, the final evaluation loss increased from 0.824 to 1.140 on the test set. The fact that the training loss tends to zero after 10 epochs, while the evaluation loss is higher than what it was at 5 epochs, indicates that the model overfits on the training data and is not able to generalize with a higher number of epochs. For this reason, the number of epochs have been kept to 5 for the remaining BERT experiments.

Results of Varying the Batch Size

After the initial experiments were run with a constant batch size of 16, the same model was run with different batch sizes under the optimal settings – a learning-rate of 3e-5 and the number of epochs set to 5. The results obtained under these settings with different batch sizes can be found in table 5.4. There seems to be no indication that a higher or lower batch size is better or worse. For this reason, the remaining experiments were run with a batch size of 16 in order to keep the

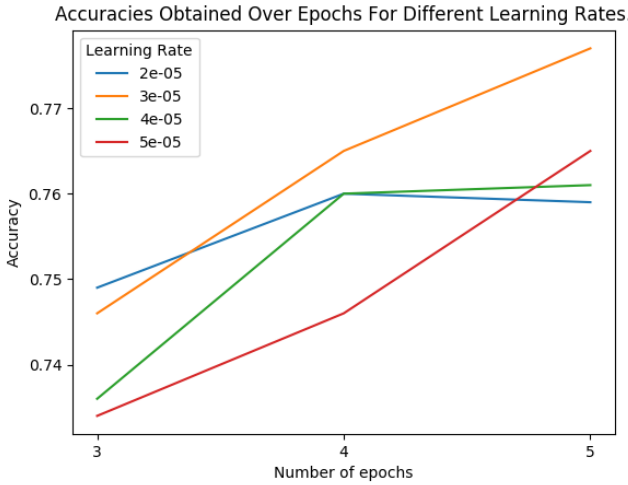

Figure 5.1: A plot showing the accuracies obtained by BERT on the TOEFL11 test set with different learning rates and total number of epochs.

Batch Size	**1**	**2**	**4**	**8**	**16**
Accuracy	0.770	0.747	0.770	0.757	0.777
Macro F_1	0.769	0.746	0.769	0.757	0.777

Table 5.4: Table of the different accuracies and F_1 scores obtained by BERT-base on the TOEFL11 test set for different batch sizes.

training time as low as possible, without encountering memory issues or reducing model performance. Additionally, the F_1 score is reported. The F_1 score of the model is typically almost identical to the accuracy of the model. This turned out to be consistent across most experiments run, which agrees with the findings from the 2017 Shared NLI Task [Malmasi et al., 2017].

BERT-Large Results

As described in the experimental plan, an experiment running BERT-large with a batch size of 1 was also carried out. BERT-large was trained and evaluated on the TOEFL11 data set, using the best performing learning-rate and number of epochs found for BERT-base, namely 3e-5 and 5, respectively. Unfortunately,

under these settings, BERT-large was not able to converge. After 5 epochs, BERT-large outputted the same probability for each L1 for all test cases, causing the model to predict the same label (Korean) for all cases. As the test set has the exact same number of examples per language, this resulted in a final accuracy equivalent to the random baseline of $1/11 = 0.091$.

In order to combat this, BERT-large was run with both a smaller and larger learning rate in order to see how the model learns. With a larger learning-rate of 5e-5, the model was still not able to produce any useful output after 5 epochs. With a learning-rate of 2e-5 however, the model produced a final accuracy on the TOEFL11 test set of **0.759** and an equivalent F_1 score. BERT-base with a batch size of 1 obtained a final accuracy of **0.770**. In other words, BERT-large does not seem to provide any significant benefit over BERT-base for the TOEFL11 data set after 5 epochs and a batch size of 1.

5.3.3 Results of Experiment 1b

The results of running BERT on the Reddit-L2 in-domain scenario can be found in table 5.5. After downsampling the data set and splitting all documents into sub-documents of max 512 tokens, there were roughly $18,300$ training examples and 1943 test cases per fold. Both sub-chunk and chunk/document accuracy is reported. After recombining all sub-chunks, there was an average of 500 test chunks per fold. The final average accuracy across the ten folds was **0.805** – a substantial improvement over the current state-of-the-art accuracy of 0.690 reported by Goldin et al.. When evaluating on each individual sub-chunk, the final accuracy drops to 0.651. This indicates that the model predicts parts of the chunks incorrectly, but performs well on the documents/chunks as a whole using majority vote. The model seems to be performing rather consistently across different folds.

5.3.4 Results of Experiment 2

The results of running BERT in the out-of-domain Reddit-L2 scenario can be found in table 5.6. The final average over random 10-fold cross-validation was **0.502** – a clear improvement over the single SVM baseline of 0.4. The result is also a substantial improvement over the current state-of-the-art of 0.362 obtained in Goldin et al. [2018]. Additionally, it should be mentioned that the train-test split in the out-of-domain scenario might be regarded as unfavorable. After downsampling, the 10% out-of-domain chunks used for testing outnumber the 90% in-domain chunks used for training. In fact, most folds had between $16,000$ and $16,500$ training examples, but somewhere in between $20,000$ and $21,000$ test cases. It is common to have the training set be between 5 and 10 times as big as the test set, but with the out-of-domain scenario the training set is around 0.785

Fold number	Sub-Chunk Accuracy	Chunk Accuracy
1	0.682	0.850
2	0.636	0.824
3	0.646	0.771
4	0.645	0.825
5	0.656	0.822
6	0.639	0.810
7	0.668	0.780
8	0.630	0.802
9	0.656	0.761
10	0.657	0.801
Final average	0.651	**0.805**

Table 5.5: Table of the different accuracies and obtained by BERT on the Reddit-L2 in-domain scenario for different cross-validation folds.

Fold number	Sub-Chunk Accuracy	Chunk Accuracy
1	0.411	0.515
2	0.406	0.483
3	0.394	0.509
4	0.405	0.499
5	0.405	0.506
6	0.392	0.508
7	0.389	0.496
8	0.408	0.490
9	0.408	0.501
10	0.382	0.513
Final average	0.400	**0.502**

Table 5.6: Table of the different accuracies and F_1 scores obtained by BERT on the Reddit-L2 out-of-domain scenario for different cross-validation folds.

times as big as the test set. This makes for a rigorous evaluation scenario, as the system is tested on more cases than that which it is trained on.

Comparing the out-of-domain results with the in-domain results of Experiment 1b, BERT's accuracy drops from 0.808 to 0.502, a $1-(0.502/0.808) = 37.9\%$ relative drop in accuracy. This, as opposed to the $1-(0.362/0.690) = 47.5\%$ relative drop in accuracy obtained in Goldin et al. [2018], indicates that BERT might be more robust than logistic regression when tested off-topic.

5.3.5 Results of Experiment 3

Next follows the experiments of using a meta-classifier and an ensemble of meta-classifiers on both the TOEFL11 and Reddit-L2 data sets. First, the results of each base-classifier trained on a different feature type are presented. Next, the results of using different meta-classifiers based on the outputs of the base classifiers are reported, both with and without BERT. Following these results are the results of using an ensemble of meta-classifiers. First, the results obtained on the TOEFL11 data set are reported, followed by the Reddit-L2 results.

	SVM				FFNN			
Feat/ Max	5000	10000	30000	inf	5000	10000	30000	inf
CHAR2	0.549	0.549	0.549	0.549	0.566	0.561	0.564	**0.569**
CHAR3	0.700	**0.704**	0.703	0.703	0.726	0.735	**0.743**	0.733
CHAR4	0.728	0.744	**0.759**	0.758	0.735	0.758	0.789	**0.795**
WORD1	0.715	0.728	**0.731**	0.727	0.732	0.755	**0.761**	0.757
WORD2	0.633	0.695	**0.731**	0.729	0.677	0.715	0.766	**0.775**
WORD3	0.517	0.570	**0.608**	0.590	0.507	0.578	0.647	**0.655**
LEMMA1	**0.712**	0.707	0.705	0.705	0.720	0.743	**0.745**	0.742
LEMMA2	0.667	0.699	0.740	**0.744**	0.665	0.745	**0.763**	**0.763**
FUNC1	0.405	0.405	0.405	0.405	**0.418**	0.415	0.401	0.405
FUNC2	0.475	0.482	**0.485**	**0.485**	0.446	0.465	0.476	**0.482**

Table 5.7: Table showing the accuracies obtained by each base-classifier for different features on the TOEFL11 test set. The maximum number of features vary between 5000, 10,000, 30,000 and no limit.

Results of Individual Base Classifiers on TOEFL11

The accuracies obtained by each base classifier for different feature types on the TOEFL11 test set are included in table 5.7. All classifiers were run with the TF-IDF weighted representation of each feature-type, with the maximum number of

features per example capped at 5000, 10,000, 30,000 and no limit. Furthermore, a visual representation of the different feature types for the SVM base classifier can be seen in figure 5.2, and the FFNN in figure 5.3. As can be read from the results, the single best performing feature type using both SVM and FFNN was character 4-grams. The SVM model seems to favor feature vectors of size 30,000, while the FFNN tends to perform better with no feature limit. For this reason, in the following experiments, the max features size has been set to 30,000 and infinite for the SVM and FFNN, respectively. It is hypothesized that the FFNN is better at filtering out the noise obtained by including more TF-IDF features, while still finding useful information when using more than 30,000 features.

Word 3-grams perform surprisingly bad when compared to word 1- and 2-grams. Additionally, as related work suggests, the content-independent function word features perform far worse than the content-dependent features.

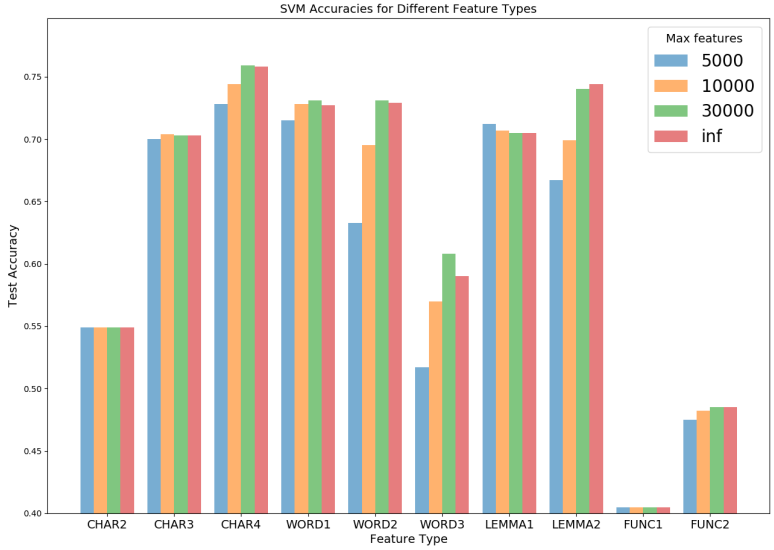

Figure 5.2: A bar diagram showing the different accuracies obtained by a single SVM model for different feature types and maximum number of features values on the TOEFL11 test set.

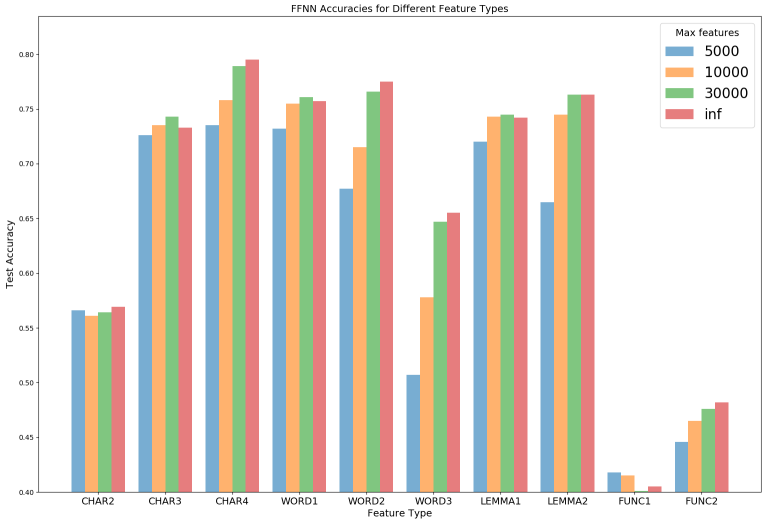

Figure 5.3: A bar diagram showing the different accuracies obtained by a single feed-forward neural net for different feature types and maximum number of features values on the TOEFL11 test set.

Meta-Classifier Results

The result of training a meta-classifier on the continuous probability outputs of the 10 base classifiers from the previous experiment can be found in table 5.8. The inclusion of BERT means appending the *n_classes* logit outputs of BERT to the training and test data. Experiments were also carried out normalizing BERT's output using softmax and raw probabilities, but using the raw logit output performed better over all. Perhaps surprisingly, FFNNs perform best both as base-classifier and meta-classifier, and the final accuracy obtained by using FFNN base classifiers and BERT with a FFNN meta-classifier is **0.853** – closing in on the current best text-only score achieved on the TOEFL11 test set of 0.882 by Cimino and Dell'Orletta.

The inclusion of BERT seems to have a significant impact on the performance of the meta-classifier. Using SVMs as base and meta-classifier, the final accuracy obtained increases from 0.756 to 0.794 – an improvement of 3.8 percentage points. The FFNN trained on the SVM base classifiers seem to reap even more benefits

Base Classifier / Meta-Classifier	SVM	FFNN
SVM	0.756	0.745
SVM + BERT	0.794	0.791
FFNN	0.819	0.825
FFNN + BERT	0.838	**0.853**

Table 5.8: Table showing the different accuracies for different combinations of base classifiers and meta-classifiers. SVMs were run with a maximum of 30,000 features, while the FFNN models were trained on an unlimited number of features.

Base Classifier / Meta-Classifier Ensemble	SVM	FFNN
SVM	0.755	0.801
SVM + BERT	0.798	0.823
FFNN	0.827	0.808
FFNN + BERT	0.849	**0.851**

Table 5.9: Table showing the different accuracies for different combinations of base classifiers and ensemble meta-classifiers obtained on the TOEFL11 test set. SVMs were run with a maximum of 30,000 features, while the FFNN models were trained on an unlimited number of features.

when including BERT, increasing from 0.745 to 0.791 – an improvement of 4.6 percentage points. The best final classifier also goes from an accuracy of 0.825 to 0.853 – a 2.8 percentage point improvement.

Ensemble of Meta-Classifiers Results

The results of training an ensemble of meta-classifiers can be found in table 5.9. Surprisingly, the results of running an ensemble of meta-classifiers is similar to just using a single meta-classifier. However, the inclusion of BERT causes a significant increase of 4.3 percentage points for the best performing ensemble. Experiments using LDA as the meta-classifier were also carried out, but provided disappointing results, a lot lower than the best single base classifier. It is hypothesized that this low performance is due to the issue of collinearity of the features used. Using a more diverse feature set might have made LDA a more viable candidate, both as a single meta-classifier and in an ensemble. As no improvement was observed using an ensemble of meta-classifiers, the remaining experiments were carried out using a single FFNN meta-classifier.

Results of Meta-Classifiers on Reddit-L2

Using the stack which performed best on the TOEFL11 data set – that is, the FFNN meta-classifier trained on the outputs of the base FFNN classifiers and BERT – the same setup was run on the Reddit-L2 in- and out-of-domain scenarios.

Meta-Classifier In-Domain Results Running the same 10 folds as the previous in-domain experiment, the final ensemble, **without** BERT, obtained an accuracy of **0.765** and an F_1 score of 0.764. Running the same ensemble **with** BERT, over the same 10 folds, yielded an accuracy of **0.818**, and an F_1 score of 0.815. In other words, BERT provides the ensemble with an absolute boost of 5.3 percentage points. Compared to the accuracy of 0.805 obtained by BERT alone on the same task in Experiment 1b, the ensemble provides BERT with an overall increase of 1.3 percentage points. Based on this, it might seem like BERT is doing most of the heavy lifting in the ensemble in the in-domain scenario, though BERT does receive a minor performance boost when used together with the ensemble.

Meta-Classifier Out-of-Domain Results The final average accuracy over the same 10 out-of-domain folds which were used in Experiment 2 obtained by the meta-classifier was **0.452**, without using BERT. Including BERT in the ensemble yielded an average accuracy of **0.529** and an F_1 score of 0.530 – an improvement of 7.7 percentage points as opposed to not using BERT. The results seem comparable to the in-domain results, as the final meta-classifier accuracy is slightly higher than the accuracy obtained by using BERT alone – from 0.808 to 0.815 in the in-domain scenario – and 0.502 to 0.529 in this case. The latter is an increase of 2.7 percentage points.

5.3.6 Results of Experiment 4

Experiment 4, as described in section 5.1.5, was concerned with using as much data as possible from the out-of-domain part of the Reddit-L2 data set, and test BERT on the entire in-domain part. After applying the construction described in section 5.1.5, the data set consisted of $1,491,198$ training examples (sub-chunks) and $282,385$ test sub-chunks. After recombining the sub-chunks into the original chunks, there were a total of $71,716$ test chunks – far more than the roughly 500 test chunks per fold in the in-domain scenario of Goldin et al. [2018] and Experiment 1b. The final accuracy obtained by BERT on all of the $71,716$ test chunks in the in-domain Reddit-L2 data set was **0.861**. For comparison, the Naive-Bayes unigram baseline classifier from the Baseline Experiments was trained and tested on the same data, and achieved an accuracy of **0.278**. The

SVM baseline classifier was also run on the same data, but unfortunately the model was not even close to finishing training after 4 days, and was therefore cancelled after reaching the maximum time limit. It would be preferable to have the SVM baseline accuracy as well, in order to evaluate how good the final accuracy actually is.

Label	**Train. Sub-chunks**	**Test Sub-chunks**	**Test Chunks**
English	80,000	80,000	19,938
German	80,000	38,442	9930
Dutch	80,000	20,291	5240
French	80,000	15,172	3889
Polish	80,000	15,071	3814
Romanian	80,000	13,232	3271
Finish	80,000	11,129	2767
Swedish	80,000	10,469	2686
Spanish	80,000	9192	2258
Greek	71,365	7802	2030
Portuguese	80,000	7176	1747
Estonian	38,371	6156	1585
Czech	59,332	5908	1520
Italian	80,000	5884	1470
Russian	55,948	5248	1391
Turkish	59,089	5039	1317
Bulgarian	40,792	4291	1067
Croatian	49,558	4121	1057
Norwegian	80,000	4068	1056
Hungarian	49,333	3930	1040
Lithuanian	47,559	3710	1016
Slovenian	24,787	3108	869
Serbian	35,064	2946	758
Total	**1,491,198**	**282,385**	**71,716**

Table 5.10: Table showing the number of sub-chunks available in the Reddit-L2 in-domain data set for different labels. English has far more sub-chunks than any other label. Additionally, the number of out-of-domain training sub-chunks are included.

However, as mentioned in section 5.1.5, the in-domain test set is not fully balanced with regards to classes, so the accuracy achieved might have been artificially high. Table 5.10 shows the different number of chunks and sub-chunks per label. English clearly has the most labels, and constitutes 27.8% of the test

labels. This is expected, as all users from Ireland, UK, US, New Zealand and Australia are grouped into this label. The same principle applies for the second largest label, German, which consists of all users from Germany and Austria. However, the final macro-average F_1 score obtained was **0.847**, indicating that the model is performing well over all classes. Thus the final accuracy obtained does not seem to be artificially high, though the slightly lower F_1 score might indicate that the model has trouble with some classes. The confusion matrix of the predicted outputs on the in-domain test set can be found in figure 5.4. As can be read from the matrix, the model seems to be slightly biased towards predicting English as opposed to other labels. The model often predicts English when the correct label is French or Dutch. Interestingly, the model confuses Norwegian with Swedish, and vice versa, both of which are also mistaken for English in 4% of the cases. As for the classes with the fewest training and test instances, such as Serbian, Slovenian and Lithuanian, the model seems to have no problem with these classes as they obtain 0.86, 0.85 and 0.91 accuracy internally within each class, respectively. In fact, both Turkish and Estonian have rather few training examples, and less than 1600 test cases, but the model achieves 0.98 and 0.94 in classifying these labels, respectively.

When evaluating the same model on the same 10 folds used in the in-domain scenario in Experiment 1b, the model obtained a final accuracy of **0.902**, and an F_1 score of **0.901**. This task can considered to be easier than the task of predicting the in-domain data set as a whole, as the average number of test cases per fold in Experiment 1b were roughly 500 chunks, as opposed to the 70K test cases when testing on the entire in-domain set. However, the 10 downsampled folds from Experiment 1b are both balanced for classes and averaged over 10 runs, making the task non-trivial. The final accuracy of 0.902 obtained when trained on more data, as opposed to training on roughly $18,000$ examples per fold as in Experiment 1b, indicates that BERT thrives with more data available. This despite the fact that the out-of-domain examples used for training contain no specific topic.

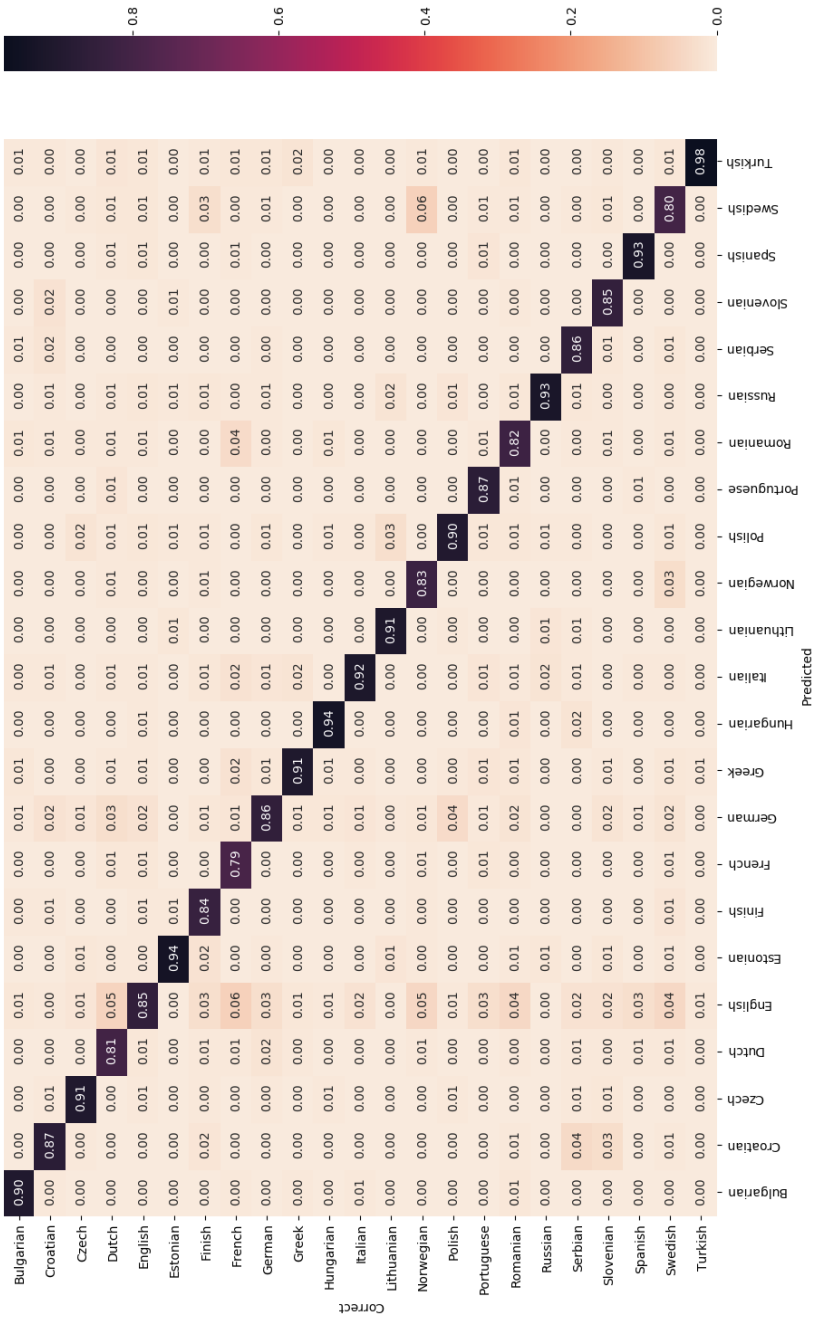

Figure 5.4: Confusion matrix of the predictions made by BERT on the entire in-domain Reddit-L2 data set, after training on the out-of-domain data. The values are normalized to be from 0 to 1, indicating how many of the model's predictions are made for which label. The matrix has been rotated for readability.

Chapter 6

Evaluation and Conclusion

Now that the results have been presented, the following chapter is dedicated to evaluating the findings of the results in light of the initial Goal and Research Questions. Furthermore, a discussion section is dedicated to the discussion of the implications of the results, as well as discussing additional questions which might have arisen based on the findings. Finally, the contributions of this Master's Thesis are summarized, and the Thesis is ended with suggestions for future work in light of the findings.

6.1 Evaluation

Looking back to the introduction, this Master's Thesis was written with a single goal in mind: *Explore how attention-based architectures can be used to improve performance on the task of Native-Language Identification.* This section will first evaluate the initial Research Questions, and finally evaluate whether the answers to the Research Questions have contributed to reaching the overall Goal.

6.1.1 Evaluation of Research Questions

Research Question 1 *How do attention-based systems perform compared to the current state-of-the-art with regards to the task of NLI?*

Experiment 1a and 1b were directly designed to answer Research Question 1. The results of Experiment 1a showed that BERT-base with the standard hyper-parameters and a learning-rate of 3e-5, was able to obtain an accuracy of **0.777** on the TOEFL11 test set, beating all the initial baselines of the Baseline Experiments. However, comparing this result to the current state-of-the-art on the task

with an accuracy of 0.882 [Ircing et al., 2017], the single BERT model is not able to compete. In fact, as the results of the individual base classifiers in Experiment 3 showed, a single FFNN trained on TF-IDF weighted character 4-grams achieved an accuracy of 0.795, beating BERT by 1.8 percentage points. Thus BERT's performance is not at the level of the state-of-the-art on the TOEFL11 test set.

Experiment 1b, however, showed that BERT was able to obtain a final average accuracy of **0.805** over 10-fold cross-validation on the same Reddit-L2 in-domain scenario as in Goldin et al. [2018]. This is a clear improvement over the previous state-of-the-art of 0.690, by 11.2 percentage points. It should, however, be mentioned that the previous state-of-the-art results were obtained during a feature exploration of the data using logistic regression. In other words, the previous state-of-the-art was not a very demanding accuracy to beat. In fact, the Baseline Experiments showed that a single SVM trained on the TF-IDF unigram scores of each document obtained an accuracy of 0.716 and also beats the original state-of-the-art score. Regardless of the lacking demands of the state-of-the-art, an accuracy of **0.805** for a 23-way classification task is a clear improvement over the random baseline of 0.049.

Furthermore, Experiment 4 showed that when BERT was trained on 1.4 million out-of-domain training sub-chunks, and evaluated on the same 10 folds as the original in-domain scenario, the model was able to obtain an impressive accuracy of **0.901**. Using more data helped improve the results obtained by BERT by almost 10 percentage points, even though the training data contained a huge selection of different topics. This accuracy is surprisingly high, and raises questions as to whether the in-domain data overlaps with the out-of-domain data. The loading of the data was separated in two distinct folders, however, so the only way this is possible is if the original data set contains overlap. Another possibility is, as some of the users have entries both in- and out-of-domain, that the user has mentioned their username in the texts, leaking information from the training to test data. However, it is not very likely that a user would be mentioning their own username, and if so, this should be the case for very few examples.

Additionally, when evaluated on the entire in-domain test set as a whole (70K test cases versus roughly 500 test cases for each fold in the previous in-domain scenario), BERT was able to obtain an accuracy of **0.861** – even higher than when trained and tested in the smaller in-domain scenario. Unfortunately, the latter is not directly comparable to any related work, as the entire in-domain part of the data set has never been used for testing before. However, it is expected that the latter task is more demanding than the initial in-domain task, as it contains 140 times as many test cases as each fold in the original in-domain test scenario. The performance of BERT with more data available indicates that

BERT's performance increases with more data, regardless of what topics it is trained on.

Based on the results obtained, a question which arises is why BERT performs better on the Reddit-L2 in-domain test set than it does on the TOEFL11. As the Reddit task contains more than twice as many labels, one would expect the task to be harder for the model than training and testing on the TOEFL11 data set. This question will be discussed further in the discussion section.

Research Question 2 *How robust are attention-based systems when tested on different topics than that which the systems are trained on?*

Experiment 2 was directly designed to answer Research Question 2. Comparing the out-of-domain results obtained in Experiment 2 with the in-domain results of Experiment 1b, BERT's accuracy dropped from 0.808 to 0.502, a 37.9% relative drop in accuracy. Compared to the current state-of-the-art, which drops from 0.690 to 0.362 – a 47.5% relative drop in accuracy – BERT seems to be more robust than logistic regression when tested on different topics than that which it was originally trained on. Again however, the original state-of-the-art was not too demanding. Furthermore, it should be noted that the out-of-domain task from Goldin et al. [2018] is a quite demanding machine learning task, as the test cases outnumber the training instances by roughly 25%. The lacking performance of both the baselines, BERT, meta-classifiers and the previous state-of-the-art, might be more of a testament to the imbalance of the task, rather than the task of training and testing on different topics.

Furthermore, as discussed under the evaluation of Research Question 1, when training BERT out-of-domain and testing in-domain, the model produces very promising results. The model seems to have no problem with being trained on different topics than it is tested on, as long as there is enough data available. A problem with Experiment 4, however, is that the testing is done in-domain. Being tested in-domain should be an easier evaluation scenario for the model than if it was tested out-of-domain. In retrospect, the model should also have been tested out-of-domain, on chunks it has not been trained on. One idea would be to take the model which was trained on the out-of-domain data, and test on remaining examples which were not present in the training data. This would not be possible, however, as for 11 of the 23 languages, all out-of-domain chunks were used for training. Using the remaining chunks of the out-of-domain data would produce a small and highly unbalanced test set, missing half of the classification labels. Alternatively, cross-validation or a similar technique could have been applied in order to make the results complete. However, as a single model took 6 days to train, a 5- or 10-fold cross-validation scheme would simply not have been feasible with the time and resources available. Regardless, testing the model on much more in-domain data should still be a testament to how the model performs when

testing on different topics than that which it is trained on.

Research Question 3 *Can attention-based systems, in combination with techniques used in the current state-of-the-art, improve performance on the task of NLI?*

Experiment 3 was directly designed to answer Research Question 3. Experiment 3 shows that the inclusion of BERT in traditional meta-classifiers or ensembles made a significant contribution to the performance of the system. The meta-classifier experiments showed that a FFNN, trained on the outputs of 10 base FFNNs with different feature types, produced an accuracy of 0.825 on the TOEFL11 test set alone. With the inclusion of the logit outputs of BERT, however, this accuracy increased to 0.853, an improvement of 2.8 percentage points. For the weaker performing meta-classifiers and ensembles, the inclusion of BERT was shown to increase the system's performance with close to 5 percentage points. The results also showed that the best ensembles and meta-classifiers performed better than BERT alone on the TOEFL11 test set. For the Reddit-L2 in-domain scenario, the increase in performance with inclusion of BERT was as much as 5.3 percentage points, and 7.5 percentage points for the out-of-domain scenario. Empirically, the combination of traditional classifiers and BERT can be used to produce better results than any of the individual classifiers.

Experiment 3 brought up some interesting questions. First of all, why did the word 3-grams perform worse than the word 1 and 2-gram features for both base classifiers? Furthermore, as opposed to the findings of the literature review, why did the FFNN generally obtain higher accuracies than the SVM? Even more important is the question of why the ensemble of meta-classifiers showed no increase in performance as opposed to using a single meta-classifier. Lastly, why did the promising LDA-classifier, which Malmasi and Dras [2018] reported to be the best meta-classifier, not give good results? All of these questions will be discussed in the following discussion section.

6.1.2 Evaluation of the Main Goal

With background in the results obtained when answering the three Research Questions, an exploration of how an attention-based system can be used to improve performance on the task of NLI has been carried out, which completes the main Goal of this Master's Thesis. The experiments show that the attention-based BERT architecture was *not* able to give state-of-the-art results on the TOEFL11 test set alone, but when combined with traditional classifiers, BERT increased the accuracy from 0.825 to **0.853** – closing in on the current state-of-the-art of 0.882. Using more features and other state-of-the-art techniques might have increased these results further. Furthermore, the attention-based system

was shown to produce a state-of-the-art accuracy of **0.902** on the cross-validation Reddit-L2 in-domain scenario, without training on any in-domain examples. Additionally, BERT gave a state-of-the-art accuracy of **0.502** on the out-of-domain Reddit-L2 scenario, which was further increased to **0.529** when including BERT as a base classifier for a meta-FFNN classifier. BERT was also shown to obtain an accuracy of 0.861 and an F_1 score of **0.847** when tested on the entire, unbalanced Reddit-L2 in-domain test set, without ever seeing an in-domain example during training. The results show that BERT and attention based systems do have merits for the task of NLI. More specifically, the findings indicate that deep learning is still beaten by traditional classifiers on the TOEFL11 test set. However, with much more data from authors who are proficient in English now available in the form of the Reddit-L2 data set, deep learning and attention-based systems can be viable candidates for solving the task of NLI.

6.2 Discussion

6.2.1 Discussion of Meta-Classifiers

A peculiar trend observed in the base classifier experiments of Experiment 3 was that the FFNN tended to favor no limit of the number of features available. Conversely, Ircing et al. [2017] report that their FFNNs did not receive any improvement when using more than $30,000$ features. It is hypothesized that as the increase was minuscule, it was considered to be non-significant. This probably because training time from $30,000$ features to unlimited increases dramatically, without providing much of a performance boost.

The next observation made was that the base classifiers trained on word 3-grams performed far worse than the smaller word grams. It is hypothesized that this might be due to the nature of TF-IDF weighting, as there are many more possible word 3-grams in the vocabulary, yet they might not be very unique with regards to documents. This can cause both a low TF and IDF score, making many of the features similar, resulting in less discriminatory power for the model. A different normalization scheme might have been favorable for word n-grams with $n > 2$.

The next interesting observation made was that the FFNNs tended to perform better than SVMs, despite that most related literature use SVMs. This is probably due to the lack of hyper-parameter tuning of the SVMs. As mentioned in section 5.2.2, the hyper-parameters of the SVM were set to be the default parameters of Sklearn. The intention was to mimic the SVMs of Malmasi and Dras [2018] as closely as possible, but as the article provides little information about the hyper-parameters used, the default parameters of Sklearn were used. One problem might have been the one-vs-rest vs the one-vs-one hyper-parameter.

Whereas Malmasi and Dras [2018] used the one-vs-rest approach for all experiments, Sklearn forces this hyperparameter to be one-vs-one when SVMs are used in a multi-class setting. This, as well as the lack of experimentation with other hyper-parameters might explain why the SVMs did not perform as well as expected. It should also be noted, however, that Ircing et al. [2017] had great success in applying FFNNs on the TOEFL11 data set. The fact that SVMs are the most popular classifier for NLI does not exclude FFNNs as a viable candidate.

Furthermore, why did LDA perform far worse than the other meta-classifiers, when Malmasi and Dras [2018] found it to be the best? The answer for this question lies in collinearity. LDA does not handle collinear variables well, which the outputs of the base classifiers are expected to be, as the output of the different labels are highly correlated. After consulting the original authors about the collinearity of the LDA classifier, Shervin Malmasi responded:

> *"For LDA, I recall that the feature representation was very important. This depends on what you are using as as base classifier. If it's a max-margin method (e.g. SVM), then you can change the distance values for each class to probabilities using something like Platt scaling. You can also try other types of feature scaling, e.g. min-max scaling or z-scores.*
>
> *The collinearity will not be completely eliminated. The amount of diversity in the feature space depends on the inputs you use. You can trying having a good mix of syntactic and lexical features as base classifiers to increase diversity."*

The first part is taken into consideration by Sklearns probability-outputs. However, applying the normalization schemes suggested might have helped. Using a more diverse feature set for the base classifiers might have made a great difference in combatting the collinearity of the training-inputs. In fact, the only syntactical feature used was function word grams – the rest were lexical features. Increasing the number of syntactic features might have increased LDA's performance.

Next, why does an ensemble of meta-classifiers not perform better than a single meta-classifier? Contrary to the results of Malmasi and Dras [2018], there was no clear gain in performance when applying a single meta-classifier as opposed to using a bagging ensemble of meta-classifiers. This is interesting, as the experiments and setup are quite similar. However, looking more closely at the results of Malmasi and Dras [2018], the increase in accuracy when going from a single meta-classifier to an ensemble of meta-classifiers is only present for the LDA meta-classifier. For instance, the final SVM meta-classifier obtains 0.852 on the TOEFL11 test set, and the final meta-ensemble of SVM classifiers obtains the exact same accuracy. The same applies to the two other regressor meta-classifiers. In other words, the benefit of applying an ensemble of meta-classifiers in Mal-

masi and Dras [2018] was only present for the LDA. Thus, the lack of increase in performance when using an ensemble of meta-classifiers as opposed to using a single one is actually expected for the SVM and FFNNs used. It is possible that the expected increase might have been present if an LDA meta-classifier was used, given that the previous issues mentioned were fixed.

Finally, it would have been interesting to incorporate some of the systems which provide state-of-the-art accuracies on their own in the ensembles used. For instance, training the string-kernel models used in Ionescu and Popescu [2017] and the stacked SVM-classifiers of Cimino and Dell'Orletta [2017], and introducing them to the ensemble might have further increased the final performance of the system.

6.2.2 TOEFL11 Versus Reddit-L2 Results

Why does BERT perform better on the Reddit-L2 in-domain data than the TOEFL11 test set, even though the latter only contains half the number of possible labels? The reasons for this could be many. First of all, the Reddit-L2 contains text written by authors proficient in English, while TOEFL11 contains text from learners of English. The texts written by the more proficient English writers might be closer to the texts which BERT was pre-trained on – English Wikipedia. A hypothesis is that the TOEFL11 data set contains many more spelling mistakes than what the Reddit-L2 data set does. This both due to the higher proficiency of the Reddit-L2 users, but also because when the Reddit-L2 documents are written, the authors often have spellchecks and other tool available in their web-browsers. A higher frequency of misspelled words can cause the WordPiece representations used by BERT to be different from that which it was pre-trained on, and might cause odd WordPiece sequences unfamiliar to the model. Put more simply, the problem might be that BERT does not handle spelling errors too well. This issue might be mitigated by taking the pre-trained BERT models and train them further on data containing spelling mistakes, using the unsupervised classification tasks described in section 3.3.2, before fine-tuning for classification. This would provide a custom BERT model where the embeddings have been trained to include spelling mistakes.

Another reason for the performance difference might be that the Reddit-L2 data set contains the label "English", while the TOEFL11 data set does not. Having one of the labels be the same as the L2 makes part of the task a binary classification task of "native or not", which is not "pure" NLI. This makes the 23-way classification task a bit easier, and might explain why the task might not be as hard as first perceived. However, the final F_1 score produced by BERT on Reddit-L2 is still high, indicating that the model is performing well across all classes. Furthermore, the Reddit-L2 accuracies increased when evaluating on the

original chunks, not the artificially created sub-chunks. This observation could indicate that the model benefits from larger documents, and that splitting and recombining them by majority vote can be beneficial. Applying a similar scheme to the TOEFL11 data set might have improved results. For instance, BERT could have been trained with a maximum sequence length of 128, or at the sentence level, training on TOEFL11 sub-chunks and recombining in the same fashion as was done using the Reddit-L2 data. In fact, if the Reddit-L2 chunks were the same size as the TOEFL11 texts, the correct initial in-domain accuracy of Experiment 1b would have been the sub-chunk accuracy of 0.65. In which case, the TOEFL11 results would have been considered to be better. Thus document size is assumed to play a pertinent role in the performance of BERT. Further motivating the importance of text granularity is the best performing text-only system on the TOEFL11 test found in Cimino and Dell'Orletta [2017], which uses a stack of SVM-classifiers, where one SVM is trained at the sentence level, and one at the document level. The same scheme could have been applied to BERT on the TOEFL11 data, training both at the sentence level and the document level.

6.3 Contributions

The contributions of this Master's Thesis are manifold. First of all, the Master's Thesis is the first so far to apply BERT to the task of NLI. An exploration using BERT on the TOEFL11 data set has empirically shown that BERT alone is not able to compete with the traditional state-of-the-art approaches for NLI on the TOEFL11 test set under the settings used. However, a meta-classifier architecture which uses both BERT and 10 traditional classifiers trained on different features was provided, which produced an accuracy of **0.853** on the TOEFL11 test set – closing in on the current state-of-the-art of 0.882.

The Reddit-L2 experiments showed that BERT can produce state-of-the-art results on the novel Reddit-L2 data set, both in- and out-of-domain. This with great results, and as much as **0.902** accuracy on the 10-fold cross-validation for the Reddit in-domain scenario, as opposed to the previous best of 0.690 – a 21.2 percentage point improvement. The Reddit-L2 experiments also provide a simple, yet effective scheme for dividing large documents for BERT, based on heuristically splitting training documents into sub-documents, and using a majority vote to recombine the predictions of the model.

An exploration of BERT's robustness when tested out-of-domain was also provided, showing that the model is more robust to topic differences than previous approaches on the Reddit-L2 data set. This both in form of BERT alone, achieving an out-of-domain accuracy of 0.502, and a meta-classifier stack, which produced a final accuracy of **0.529** – a 16.7 point improvement over the previous

out-of-domain state-of-the-art. The latter experiment also uncovered an imbalance in the out-of-domain train-test split described by Goldin et al. [2018], where the test cases typically outnumber the training instances.

Furthermore, the Thesis provides a thorough discussion as to why BERT performs so much better on the Reddit-L2 data set than the TOEFL11 data set, and presents a hypothesis that this is due to the number of spelling mistakes found in the TOEFL11 data, in addition to the document granularity of which the model is trained on.

The Thesis provides an exploration of ensembles and meta-classifiers for NLI, and shows that the inclusion of BERT in these classifier stacks can improve the final accuracy with up to 10 percentage points, depending on the task and base-classifiers used.

Finally, a large experiment with 1.4 million, out-of-domain training documents is provided, for which BERT obtains an F_1 score of 0.847 on the *entire* Reddit-L2 in-domain data – a test set which contains more than 140 times as many test cases as the original in-domain scenario. The final experiment was also done by training out-of-domain on a wide range of topics, displaying the model's robustness and indifference to topics given large amounts of data.

6.4 Future Work

As BERT obtained such promising results on the Reddit-L2 in-domain data set, future work would include looking into how to increase the accuracy on the TOEFL11 test set, as this still is the standard data set to use for NLI. The main focus of the future work would be to look into continuing the pre-training of BERT on parts of the TOEFL11 data set, in order to learn embeddings of spelling errors and other attributes of the data, before training for classification. Next, looking further into how the TOEFL11 data is used with BERT would be of importance. It would be interesting to train BERT on TOEFL11 using smaller training examples, for example splitting documents into sub-documents with a maximum sequence length 128, or at the sentence level, and then recombining the predictions in the same fashion as was done for the Reddit-L2 sub-chunks. Using smaller text pieces would also allow for running BERT-large with a shorter sequence length and larger batch sizes, without requiring more computing power. With more power available, running BERT-large with a sequence length of 512 would also be feasible. In addition, it would be interesting to carry out more experiments using both the cased and uncased versions of BERT, in order to see whether this impacts the performance of the model. Furthermore, opening up the attention mechanism and looking at what parts of the input sequence the model is paying attention to could be used to give new insights in the field. This sort of analysis would also be useful in educational platforms, and the findings

could potentially be used to help educators of different languages.

Regarding the ensembles and meta-classifiers, future work would be to include other state-of-the-art systems, such as string-kernel SVMs and more features, which could possibly improve results further.

Additionally, with the novel Reddit-L2 data set available, deep learning models are again a viable option for NLI. For this reason, a new Shared Task for NLI using the Reddit-L2 data or equivalent could lead to interesting advances in the field. Producing a common train-test split for the Reddit-L2 data set would help the community explore novel approaches with more data available, and make the results comparable – just like the TOEFL11 data set did in 2013. Additionally, future work would also include training and testing on the Reddit-L2 data set when excluding all English examples. This would remove the "native or not" element of the task, making it a more pure NLI task. Removing English would also make the Reddit-L2 results more comparable to TOEFL11 results.

The size of the Reddit-L2 data set could also be increased by further web-scraping. Not only would this yield even more data, but it would help balance out the label imbalances the data set currently has. This would also allow for the creation of a more balanced out-of-domain scenario, where the training examples are not outnumbered by the test examples. Furthermore, future work would include looking into how the Reddit-L2 data could be pre-processed for better results. For instance, as the data is created online, chunks often contain URLs etc., which might function as noise to the model. Performing a proper exploration of the data and pre-processing could help further increase the quality of the data.

Finally, training and testing cross-corpus from the Reddit-L2 data to the TOEFL11 data would possibly expose new insights in the field of NLI. For instance, training on the Reddit-L2 data and testing on the TOEFL11 test set would yield results which would be directly comparable to previous work done on the TOEFL11 data set. This would enable a closer look at the role of misspelled words and data size in NLI, both for attention-based systems and traditional classifiers. The main obstacle for such a cross-corpus experiment would be that the Reddit-L2 data set currently does not contain all the languages present in the TOEFL11 data set. However, this problem could be solved by the aforementioned web-scraping extension of the Reddit-L2 data set.

Bibliography

Alammar, J. (2018). Visualizing a neural machine translation model (mechanics of seq2seq models with attention). `https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/`. Accessed: 2019-03-12.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Blanchard, D., Tetreault, J., Higgins, D., Cahill, A., and Chodorow, M. (2013). Toefl11: A corpus of non-native English. *ETS Research Report Series*, 2013(2):i–15.

Brooke, J. and Hirst, G. (2013). Using other learner corpora in the 2013 NLI shared task. In *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 188–196.

Cimino, A. and Dell'Orletta, F. (2017). Stacked sentence-document classifier approach for improving native language identification. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 430–437.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Goldin, G., Rabinovich, E., and Wintner, S. (2018). Native language identification with user generated content. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3591–3601.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

Huq, K. T., Mollah, A. S., and Sajal, M. S. H. (2018). Comparative study of feature engineering techniques for disease prediction. In *International Conference on Big Data, Cloud and Applications*, pages 105–117. Springer.

Ionescu, R. T. and Popescu, M. (2017). Can string kernels pass the test of time in native language identification? *arXiv preprint arXiv:1707.08349*.

Ionescu, R. T., Popescu, M., and Cahill, A. (2014). Can characters reveal your native language? A language-independent approach to native language identification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1363–1373.

Ircing, P., Svec, J., Zajic, Z., Hladká, B., and Holub, M. (2017). Combining textual and speech features in the NLI task using state-of-the-art machine learning techniques. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 198–209.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koppel, M., Schler, J., and Zigdon, K. (2005). Determining an author's native language by mining a text for errors. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 624–628. ACM.

Lei Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444.

MacDonald, M. C. (2013). How language production shapes language form and comprehension. *Frontiers in Psychology*, 4:226.

Malmasi, S. and Dras, M. (2018). Native language identification with classifier stacking and ensembles. *Computational Linguistics*, pages 1–70.

Malmasi, S., Evanini, K., Cahill, A., Tetreault, J., Pugh, R., Hamill, C., Napolitano, D., and Qian, Y. (2017). A report on the 2017 native language identification shared task. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 62–75.

McCallum, Andrew, N. and Kamal (1998). A comparison of event models for Naive-Bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Opitz, D. and Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198.

Rabinovich, E., Tsvetkov, Y., and Wintner, S. (2018). Native language cognate effects on second language lexical choice. *arXiv preprint arXiv:1805.09590*.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Sasaki, Y. (2007). The truth of the F-measure.

Singer, P., Flöck, F., Meinhart, C., Zeitfogel, E., and Strohmaier, M. (2014). Evolution of Reddit: From the front page of the internet to a self-referential community? In *Proceedings of the 23rd international conference on world wide web*, pages 517–522. ACM.

Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Svozil, D., Kvasnicka, V., and Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, 39(1):43–62.

Tetreault, J., Blanchard, D., and Cahill, A. (2013). A report on the first native language identification shared task. In *Proceedings of the eighth workshop on innovative use of NLP for building educational applications*, pages 48–57.

Tetreault, J., Blanchard, D., Cahill, A., and Chodorow, M. (2012). Native tongues, lost and found: Resources and empirical evaluations in native language identification. *Proceedings of The 21st International Conference on Computational Linguistics 2012*, pages 2585–2602.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Voutilainen, A. (2003). Part-of-speech tagging. *The Oxford Handbook of Computational Linguistics*, pages 219–232.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., and Macherey, K. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Stian Steinbakken

Paying Attention to Native-Language Identification

NTNU
Norwegian University of
Science and Technology