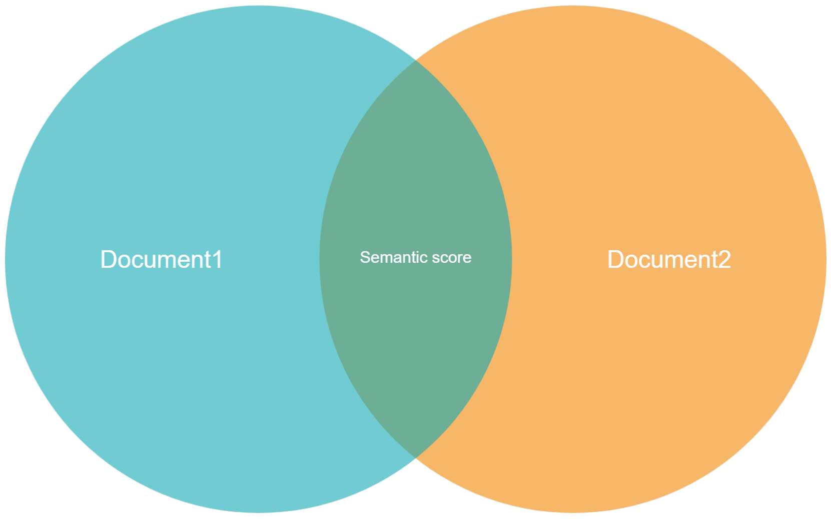
Semantic Text Similarity



Martin Björn Simon Gustafsson Oskar Skoglund
Joakim Tao David Thimren Tim Yngesjö

TDDE19 Advanced Project Course - AI and Machine Learning
Linköping University

Chapter 1

Introduction

Determining similarity between sentences is an important and fundamental task in Natural Language Processing (NLP). Being able to compare the semantic similarity between texts has many applications in a wide variety of areas. Examples of areas where text similarity is being used are plagiarism detection, search engines and customer service. This report aims to evaluate and compare different semantic text similarity algorithms based on the accuracy metric.

1.1 The problem

The text similarity problem is that given two texts t_1 and t_2 predict if the texts are semanticly similar. The texts may have different number of characters and words. To achieve this, supervised learning can be applied to the problem. The problem is a binary classification problem, i.e. the prediction can be exactly one out of two values. The goal is to construct a prediction function $h_\theta(t_1, t_2) \rightarrow \{0,1\}$ where t_1, t_2 are the two texts and θ are parameters of the predictor function.

1.2 Possible solutions

There are many different algorithms to solve the semantic text similarity problem. In this project, the following algorithms were compared and evaluated: term-frequency inverse document frequency (TF-IDF), Bidirectional Encoder Representations from Transformers (BERT), Long short-term memory (LSTM) and gated recurrent unit (GRU).

1.3 Hypothesis

The hypothesis of this project is that TF-IDF will perform worse than both LSTM and BERT. The reasoning is that TF-IDF is a very simple way to model the problem. It ignores the context of a sentence and also the sequence of words, but it will probably be a fast model that can give a simple overview of the similarity between documents. TF-IDF is a unsupervised algorithm, meaning that it disregards the ground-truth labels and only calculates similarity based on the word-vector space.

In contrast to TF-IDF, the supervised model, LSTM, will give a more accurate prediction of the similarity problem and will have a higher accuracy than the TF-IDF model. Given what previous scientific works get as accuracy when using a LSTM model in similar problem, it will most likely get a accuracy of 80-85% in our task.

Finally, the last hypothesis of the project is that the BERT model will outperform TF-IDF and slightly outperform LSTM. BERT is a more recently developed algorithm compared to LSTM and according J. Devlin et al. the model has improved the results in seven different NLP tasks [3]. Therefore it is probable that it will outperform LSTM in the text similarity task as well, most likely with a resulting accuracy of 85-90%.

Chapter 2

Method

This chapter describes the pre-processing step as well as the implementation process. The main goal of the project is implement different algorithms which are able to distinguish if two question are divergent or not. The implementation includes 1 baseline model TF-IDF and 5 algorithms LSTM(MaLSTM), BiLSTM, GRU, BERT and ALBERT. The best performing model will have a fine-tuning process and will be presented as the final model.

2.1 Dataset

To train and evaluate the accuracy of the different models implemented, a dataset of Quora Question Pairs were used [1]. The dataset is part of a publicly available Kaggle competition, with the goal to classify if two sentences(questions) have the same meaning. The set contains over 400 000 question pairs, with given ground truth for each pair labeled as 1 if the two sentences has the same meaning, and 0 otherwise. The ground truth has been labeled by human experts, and can therefore be subjective.
Some examples of question pairs from the dataset:
"How can I be a good geologist?","What should I do to be a great geologist?", 1
"What is the best travel website in spain?","What is the best travel website?", 0

2.2 Pre-processing steps

The questions in the data set are an exact replica of the questions asked on Quora which contain a lot of uninformative words and are not in the format which the models can train on. Some question might even contain misspelled words. Before training the models, a pre-processing steps is needed to both remove uninformative words which are known as stopwords, and also reformat the question.

The first step was to clean up the question. The questions were filtered using regular expression operations from python. More specifically the project used the *re.sub(pattern, repl)* function, which matches a regular expression (pattern) in the question and replaces it (repl). This was done to filter out the non-alphabetical words but also used to reformat common contraction like "*I'll* to *I will*".

After reformatting the questions using regular expression, the stopwords were filtered out using the list of stopwords from NLTK library [2] in python .

Finally there was also a fixed question length, this was implemented to reduced the amount of zero-padding needed for the shorter questions. The zero-padding was necessary

[1]https://www.kaggle.com/c/quora-question-pairs
[2]https://www.nltk.org/

since most of the models required feature vectors of the same length. The fixed question length was 100 words, this means that the shorter questions were zero-padded while the longer questions were cut off at 100 words. For the model using BERT or ALBERT the word length was set to be no longer than 35 characters long, since the model crashed when a word was too long. The limit of 35 characters was set so that all normal english words would fit, including "supercalifragilisticexpialidocious". Words longer were removed from the question.

2.3 Baseline

A classifier based on Tf-idf was used as a baseline for this project. For each sentence pair u,v, tf-idf vectors were formed

$$\begin{aligned} v &= \text{tf-idf}(t_i, d_1, D) \quad i = 1, \cdots, M \\ u &= \text{tf-idf}(t_i, d_2, D) \quad i = 1, \cdots, M \end{aligned}$$

where M is the number of words in the vocabulary. The similarity between the vectors u,v was then calculated using cosine similarity defined by

$$\text{cos\_similarity}(u,v) = \frac{u^T v}{\|v\|_2 \|u\|_2}$$

Two sentences are classified as similar if their cosine similarity exceeds some threshold, this can be written as

$$\frac{u^T v}{\|v\|_2 \|u\|_2} \geq \alpha$$

where $a \in (0,1)$. Experiments for different values of α were conducted.

2.4 LSTM,GRU, and BiLSTM

For this project the popular deep learning library, keras [3], was used to implement the LSTM and the bidirectional LSTM models. The method was implemented with a many to one implementation meaning that the model only gave one output for each sentence. This output was a vector with a length of 50. The structure of the model was two siamese LSTMs that both took one question each and then produced a vector, which would be compared to each other and than a final prediction would be produced. A very similar model which used gated recurrent units instead of LSTMs was also implemented.

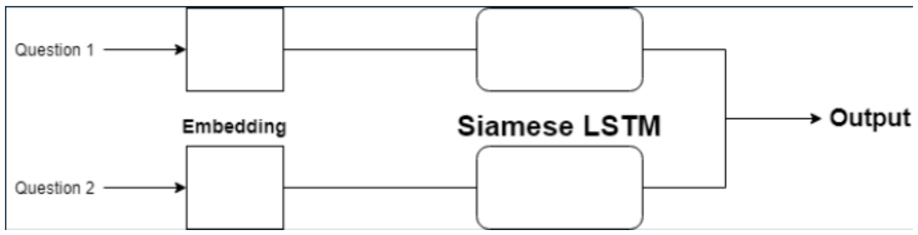


Figure 2.1: LSTM model structure

[3]https://keras.io/

The structure was used for both the LSTM, GRU and the bi-LSTM model and for both the LSTM model.

To use the sentences in the LSTM models they have to be transformed through a embedding matrix, created with word2vec, that has been pre-trained. The specific one used in this project was the Googlenews-vector-negative300.bin. Now each word can pass through these embedding matrix as one hot vectors which would yield the vector representation needed for the LSTM:s.

2.5 BERT and ALBERT

The model using BERT [3] or ALBERT [4] consists of the base version of the respective model, a bidirectional LSTM, max and average pooling, dropout, and lastly a dense layer that outputs the prediction. Unless otherwise specified, the LSTM layer has 64 units, and the dropout is set to 0.3. This project used HuggingFace transformers [8] implementation of BERT and ALBERT in Keras and Tensorflow. Before the questions are passed through the model, the two sentences were first encoded together and separated with [SEP] token using HuggingFace implementation of BERT and ALBERT tokenizer.

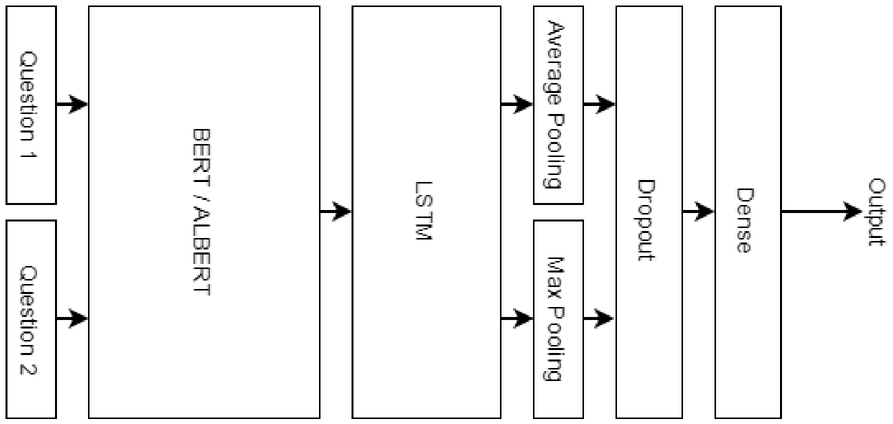


Figure 2.2: BERT/ALBERT model structure

2.6 Fine-tuning

The model with the highest accuracy was then fine-tuned to try and get a better accuracy. To fine-tune the hyper parameters of the model trial and error was used to see what gave the best accuracy. To save time during the fine-tuning, only 100 000 question pairs were used for training and 10 000 question pairs for validation. As seen in Table 3.1 the different values or length was tested of these parameters:

- Word pre-process
- Max sentence length
- Instead of long word
- LSTM size

When using *Word pre-process* the same pre-processing steps was used as in section 2.2. *Instead of long word* was used to see if the model would learn that two questions, with the *Instead of long word* characters in them, might be related.

Chapter 3

Results

This chapter presents the performance of the baseline models TF-IDF and the performance of evaluation models BERT, ALBERT, MaLSTM and BiLSTM. The performance is measured using accuracy.

3.1 TF-IDF

The projects baseline model Tf-idf achieved an accuracy of **65.2 %**. The hyper-parameter α was optimized by varying the α value. The experiments resulted in $\alpha \approx 0.6$ yields the highest accuracy, this can be seen in Figure 3.1.

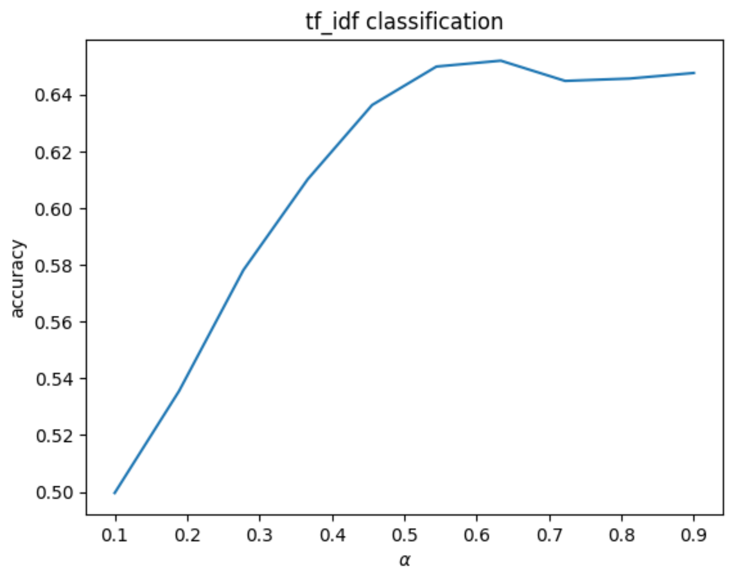


Figure 3.1: Accuracy of tf-idf for different values of α

3.2 LSTM

The projects Manhattan based LSTM model achieved an accuracy of **81.5 %** on the test data and a 86.5 % on the training. The accuracies for the train and test data over 25 epochs can be seen in Figure 3.2.

3.2.1 GRU

The GRU-based model achived a training accuracy of 82 % and a test accuracy of **80.0 %**. The accuracy during training can be seen in Figure Figure 3.2

3.2.2 Bi-directional LSTM

The Bi-directional LSTM, BiLSTM achived an accuracy of **81.3 %** on the test data and 86.3 % on the training data. The accuracies over the 25 trained epochs for both the train and test set can be seen in Figure 3.2.

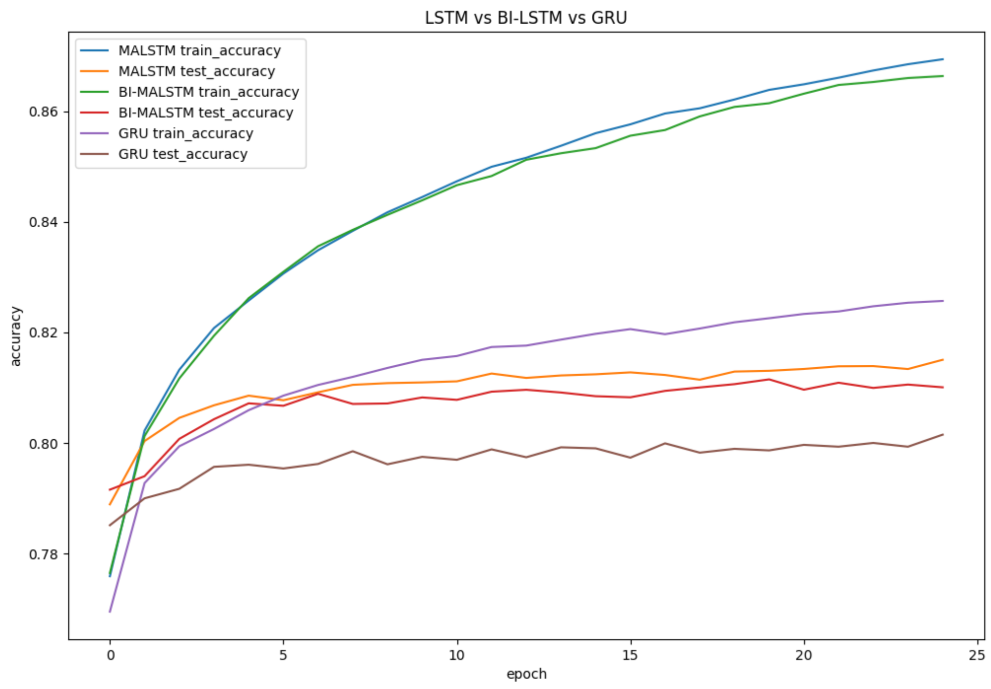


Figure 3.2: Accuracy of LSTM vs BiLSTM vs GRU model

3.3 BERT

The projects BERT model achieved an accuracy of **85.8 %** on the test data and an accuracy of 85.0 % were also achieved on the train data. The accuracies over the 10 trained epochs for both the train and test set can be seen in Figure 3.3.

3.3.1 ALBERT

The ALBERT model resulted in an accuracy of **87.2 %** on the test data and an accuracy of 91.4 % on the train set. The accuracies over the 10 trained epochs for both the train and test set can be seen in Figure 3.3.

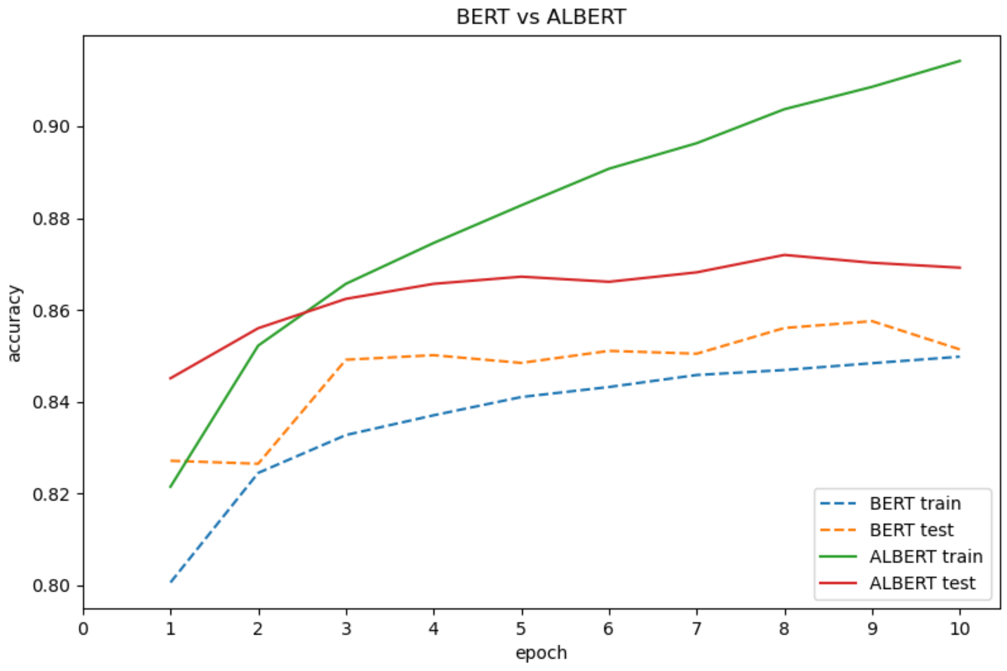


Figure 3.3: Accuracy of BERT vs ALBERT

3.4 Fine-tuning

Comparison between the test accuracies for the MaLSTM, GRU, BiLSTM, BERT and ALBERT models shows that the ALBERT model achieves the best accuracy and was thus chosen to be fine-tuned.
The results and variation of fine-tuning the hyper-parameters of ALBERT can be seen i in the Table 3.1

Table 3.1: Accuracy with ALBERT using different parameters

Word pre-process	Max sentence length	Instead of long word	LSTM size	Acc..
Yes	50	Use the long word	64	Error
Yes	50	".....”	64	0.8547
No	50	".....”	64	0.8600
No	50	"@”	64	0.8631
No	50	””	64	0.8627
No	100	"@”	64	0.8619
No	100	””	64	**0.865**
No	150	””	64	0.8645
No	100	””	32	0.8582
No	100	””	128	0.8619

3.4.1 Final ALBERT

After the fine-tuning process the final ALBERT model achived an test accuracy of **88.2 %** and a train accuracy of 91.4 %. The accuracies over 10 epochs for the final ALBERT model can be seen in Figure 3.4

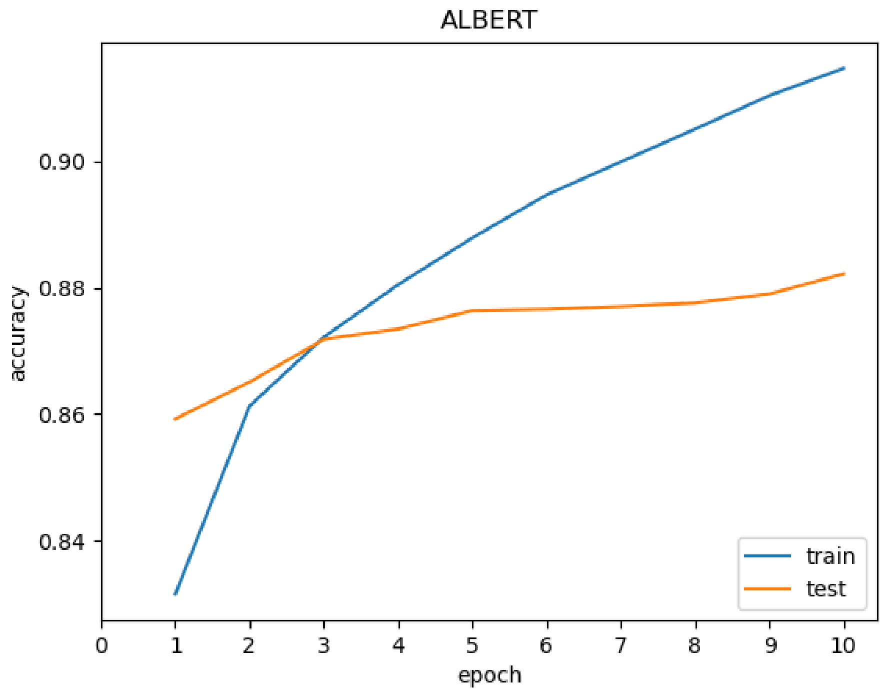


Figure 3.4: Final ALBERT

Chapter 4

Discussion

This chapter will discuss the validity of the results as well as an evaluation of the model and methods used in the project. This chapter will also include some discussion of future works.

4.1 Pre-processing

The pre-processing used in this project is the standard pre-processing used in most of project in text classification. First the removal of non-alphabetical words and expanding contractions, this is always a good idea to do to limit the scope of words the model would have to learn. After that, all stop words in the questions were removed because they don't contain any relevant semantic meaning and would only introduce noise for the model and the result would likely be worse. Lastly using a fixed length for the input sequences was done so to limit the zero padding needed for the shorter questions. Which would mean that longer questions would loose some information but would also improve the efficiency of the model by making it significantly smaller, and this is worth it because only outlier questions would exceed the max length.

What was not done but could have improved the results obtained was fixing the lemmatization for each word, for example changing driving into drive, so to reduce the number of variations for each word that means semantically the same thing. This could help the model understand some words better and improve the accuracy for the model.

4.2 Sentence length limitations

It seems like reducing the maximum sentence lengths helped most LSTM/GRU models to learn better. Since the models operate on sentences of fixed length, all sentences needs to be converted to the same length. This is done by zero-padding all the sentences to get the same length as the longest sentence. While preserving all the information about the sentences, this method introduces redundancy in the data that is fed into the models. In an attempt to make the input more information dense, a max sentence length was introduced. Although this removes some information about some long sentences the results become better. A max sentence length of about 50 words seemed to yield the best result. Why a relatively low max sentence lengths worked good is probably due to the fact the most questions in the data set are relatively short.

4.3 Results

The different models performed significantly better than the baseline (TF-IDF), which is expected. Out of the five algorithms BERT and ALBERT performed better than their RNN-based counterparts, as hypothesized. ALBERT performed the best with an accuracy of 88%. Out of the RNN-based models MaLSTM performed the best with an acccuracy of 81%. This accuracy also agrees with the hypothesis.

Comparing the results with *Semantic Textual Similarity on MRPC* [5] which is a similar problem, the models perform below the state of the art models that reach an accuracy of 93.4% using ALBERT. However the two data sets used are not the same, meaning the same results cannot be expected. In addition, the ALBERT used in our model was base model, there exist larger models (Large and XL), changes of this could result in higher accuracy.

One thing to note is that the average human accuracy is unknown, which means it is hard to tell how good an accuracy of 88% actually is. Additionally the questions have been labelled by humans and can therefore be subjective.

4.4 BERT vs ALBERT regarding dropout

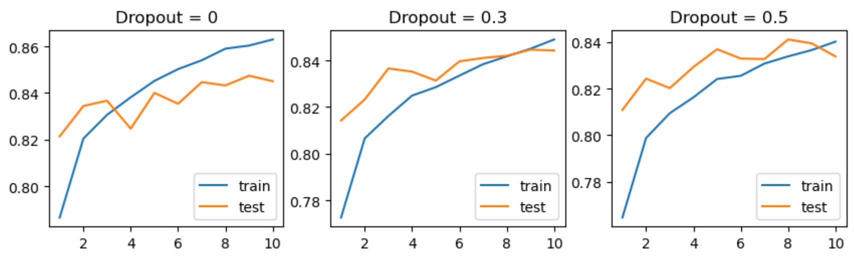


Figure 4.1: Accuracy of BERT for different values of dropout

ALBERT seems more robust compared to BERT, where BERT might rely too much on only a few parameters per output, so when some of those parameters are dropped, it loses too much information to give an accurate guess. However, as can also be seen in Figure 4.1, this does not affect the testing accuracy since the dropout is only active while training. One possible explanation of the difference between the effect of dropout on the model using BERT and the model using ALBERT, is that BERT already uses dropout inside the model during training, while ALBERT has no dropout. It could be that using the dropout in BERT plus the dropout in our model is too much for the model to make a good prediction.

4.5 Future Works

One part of the project that could be looked at in order to improve the performance is the data. The data set consists of 400 000 question pairs, but compared to some of the larger open source data sets *Blog Authorship Corpus* and *Amazon Product Data set* which consist of around 140 million data 400 000 is not a lot. To further improve the performance of the model better training would be needed thus a larger data or in this case some kind of data augmentation.

Data augmentation are techniques used to increase the amount of data by slightly modifying existing data and adding them or creation of synthetic data from existing

data [7]. There are two more well known data augmentation techniques one is GAN also known as Generative Adversarial Networks, the other is EDA, also known as easy data augmentation.
EDA is a method that increases the data by slightly modifying the data by using 4 main methods in EDA, more details can be found in [9]. GANs normally consists of two neural networks which train and competes against each other. One of the networks works as the generator which generates data that is indistinguishable from real-world data while the other network works as the discriminator to distinguish if the generated data looks like real data or not [2, 6]. More details about GAN can be found in [2, 6].

Both data augmentation method could be explored to see the advantages of both. While the EDA have a more simple approach it is also less robust, while a GAN based approach is more advanced but it gives a more robust solution.
Caccia et al. (2018) investigated language based GAN models and came to a conclusion that well-adjusted language model outperforms the GAN variants [1]. Caccia et al. (2018) also states that "GAN training may prove fruitful eventually, but this research lays forth clear boundaries that it must first surpass." [1]. But a more recent research states the opposite. The research was about GAN in text classification and introduced a new method called GAN-BERT [2]. GAN-BERT is a method which uses a semi-supervised GAN (SS-GAN) in BERT fine-tuning. More implementation details can be found in [2]. In the paper Croce, Castellucci and Basili (2020) explains that GAN-BERT was proven to systematically improve the robustness of all while not bring in any additional costs to the inference [2]. So a GAN-BERT approach for synthetic data generation would be worth investigating.

Bibliography

[1] Massimo Caccia et al. "Language GANs Falling Short". In: *arXiv e-prints*, arXiv:1811.02549 (Nov. 2018), arXiv:1811.02549. arXiv: `1811.02549 [cs.CL]`.

[2] Danilo Croce, Giuseppe Castellucci, and Roberto Basili. "GAN-BERT: Generative Adversarial Learning for Robust Text Classification with a Bunch of Labeled Examples". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 2114–2119. DOI: `10.18653/v1/2020.acl-main.191`. URL: `https://www.aclweb.org/anthology/2020.acl-main.191`.

[3] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: `1810.04805 [cs.CL]`.

[4] Zhenzhong Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2020. arXiv: `1909.11942 [cs.CL]`.

[5] paperswithcode. *Semantic Textual Similarity on MRPC*. 2021 (accessed January 09, 2021). URL: `https://paperswithcode.com/sota/semantic-textual-similarity-on-mrpc`.

[6] Pegah Salehi, Abdolah Chalechale, and Maryam Taghizadeh. "Generative Adversarial Networks (GANs): An Overview of Theoretical Model, Evaluation Metrics, and Recent Developments". In: *arXiv e-prints*, arXiv:2005.13178 (May 2020), arXiv:2005.13178. arXiv: `2005.13178 [cs.CV]`.

[7] Great Learning Team. *Understanding Data Augmentation — What is Data Augmentation & how it works?* Aug 5, 2020 (accessed Jan 05, 2021). URL: `https://www.mygreatlearning.com/blog/understanding-data-augmentation/`.

[8] The Hugging Face Team. *Transformers*. 2020 (accessed January 08, 2021). URL: `https://huggingface.co/transformers/index.html`.

[9] Jason Wei and Kai Zou. "EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks". In: *arXiv e-prints*, arXiv:1901.11196 (Jan. 2019), arXiv:1901.11196. arXiv: `1901.11196 [cs.CL]`.