Training Transformer Model for Swedish Sentiment Analysis

Mosleh Mahamud
Bachelors of Science in Computer Science
Department of Computer and System Sciences DSV
Stockholm University
Stockholm, Sweden
mosleh.edu@gmail.com

***Abstract*—Sentiment Analysis is a subfield within natural language processing. This paper looks at if a Swedish pre-trained BERT model could be leveraged to classify polarity sentiments from text and its performance compared to a convolution neural network. The specific BERT model used is the KB BERT with the help of Hugging face library and PyTorch. This paper compares test results between BERT and CNN, where the same dataset is used to compare their accuracy. The CNN is already tested in literature, and therefore only BERT fine-tuning will be implemented for this research paper. The results show that KB-BERT fine-tuning performs comparatively worse than CNN by about 30%. However, there is a caveat, if BERT is fed more data in either pre-training or fine-tuning it could prove to work significantly better than CNN.**

***Index Terms*—KB-BERT, Transformer, Sentiment Analysis, Swedish, NLP.**

I. Introduction

SENTIMENT analysis is a common NLP task which aims to classify texts to derive insights. The deriving of sentiments from texts could be useful to solve. For instance, if a company has many reviews and needs to get insights fast and accurately, deep learning algorithms could be used to achieve that. This paper will discuss how to fine-tune a BERT model to classify sentiment polarity from texts in Swedish. This fine-tuning procedure is an example of transfer learning, where pre-trained models will be leveraged to achieve this classification task. Another sentiment analyzer has been created in the past using Convolutional neural network. This paper will also try to compare the performance between the two analyzers accuracy. This fine-tuning implementation will give a controlled comparison between the BERT and CNN as the same dataset used in CNN[9] will be used fine-tune KB-BERT. Accuracy will be the most crucial aspect of how this will be compared.

A. Purpose

The purpose of this implementation project is to find out if a transformer-based language model outperforms a convolutional neural network (CNN) model when it comes to text classification such as sentiment analysis. A comprehensive description of the CNN variant is discussed in section 2 (Related Work). But more specifically, this paper is going to compare the final accuracy from both implementations. A Swedish BERT model will be used to attempt this experiment. The specific BERT model is discussed more in section **2** (Related Work)

B. Bert

Bert is essentially a language model that can solve the state of the art NLP tasks such as question answering, language inference and even contextual recommendation as this paper will explore. BERT stands for Bidirectional Encoder Representation from Transformer [2]. As the name suggests it borrows the encoder part from the transformers architecture. The model provided by Google is already pre-trained with large amounts of corpus because "language model pretraining has shown to be effective for improving many natural language processing tasks"[8]. Therefore, many tasks can be solved with the Bert model like grammar error detection [13], Text summarization [5], Sentiment analysis as this paper will look at and many more. These types of tasks can be achieved with minimal fine-tuning with datasets because there are Bert models that are already pre-trained.

1) Pre-Trained Bert: Bert is trained with two unsupervised tasks such as Masked LM(MLM) and Next Sentence Prediction (NSP). The data used to train the model is from many large unlabeled corpora. **Task 1** is about masked LM which is essentially takes sentences and "masking" 15% of the word piece in the corpora,then letting the model predict what the missing [MASK] tokens are in order to complete the sentence based upon it's context [2]. **Task 2** was about how the model is able to teach relationships between sentences in order to predict what sentences come after an inputted sentence

2) KB-Bert: The model used for this paper will be the KB-Bert model bert-base-swedish-cased (v1.1).The National library of Sweden released a pre trained bert model with 15-20 GB of Swedish texts which is equivalent to 200M sentences or 3000M tokens from various sources such as books, news, government publications etc [3]. This trained model also had the same hyper parameters published by Google

C. Transformer

Transformer is a new neural network architecture that can solve NLP tasks for example language translation. This was also published by researches on Google [12]. This was built so that the language model could perform better when it comes

to classifying longer texts without hampering performance [14]. This paper will only explore the encoder section of the architecture as it is the most relevant for BERT.

Bert is built using multiple stacks of encoder layers. These encoder layers or Transformer blocks [2] can vary as BERT base has 12 layers whereas BERT large has 24. They also have their respective "feed-forward networks 758 and 1024 hidden units respectively." [1]. The model used in for this paper uses a Bert base cased model as described in section 2.

1) Encoder: Each encoder essentially takes in a sequence of sentences and converts them into a "well enriched" continuous vector representation of the inputs. The reason for that is to find the a contextual representation of the sequence of inputs. This goes through a multi-headed self-attention mechanism to achieve that.

The encoder is essentially built with 4 central parts where they are done sequentially. Although it is worth mentioning that Multi-headed attention mechanism is a very parallel process.

Self attention mechanism is about how the algorithms "understands" the context of a sentence and focuses on the words that have the most relevance. In the encoder, the self-attention mechanism happens multiple times in parallel, thus it's called Multi-headed attention [2]. This will allow BERT to be able to map the polarity of sentences once fine-tuned with an extra layer of classifier.

II. Related Work

An implementation research was at Linné University, where positive and negative classed labelled data was used to train a convolutional neural network. Dataset used in that paper was collected from www.reco.se and www.trustpilot.com. Classification using two-class labels, positive and negative the CNN based model reached an astounding 95%. The models used for training the sentiment analyzer was based upon a convolutional neural network architecture. Around 10 CNN models were used to train and test where almost all of the model had an accuracy above 94% for binary labelled positive and negative classes. The paper also depicts that more overall data is needed for achieving high performance when classifying sentiments rather than having perfect class balance in the dataset [9].

Part of why this paper will train a Swedish sentiment analysis is because, there was one implementation using a pre-trained English BERT model that was fine-tuned with reviews from google play store, which achieved remarkable results when it comes polarity based sentiment analysis. Venelin shows an 88% accuracy on multi-label text classification with only 16000 examples containing positive, negative and neutral [11]. Additionally, a graduate thesis from Lund University did a binary sentiment text classification using different language models and found out that the KB-BERT pre-trained model outperformed any other Swedish-based language model with an accuracy of 96% ! [7]. However, it is worth mentioning that the fine-tuning data was about 21,851 reviews containing 8,112 negative reviews.

III. Method

The method aims to fine-tune the pre-trained KB-BERT model with varying sizes of a dataset. The dataset will be prepared for two stages to experiment with varying class label distribution to check its effects on the transformer model's accuracy. The fine-tuning procedure will be about making a custom classifier on top of BERT model that will allow for sentiment classification; details about the classifier layer is described on section 3 B. Then the model will be tested on a test set followed by evaluation using Accuracy, ROC AUC, F1 and precision

A. Dataset

The dataset was taken from an open-source project done by a former student in Linné university as their thesis ([9]). The dataset was originally extracted from www.reco.se and www.trustpilot.com website, which included 14,296 examples of 9667 positive and 4629 negative examples. As aforementioned, this data set was from an open-source project, and the person was contacted in person for this research to happen. This dataset will not be used for any commercial purposes but only for strict academic research intentions.
The experimentation was conducted in two stages, and therefore, the dataset was wrangled in two different ways for each experimentation stages.

For **stage one** of the experiment, nothing was changed from the original dataset that has 14,296 examples. The dataset was first split into train and validation for training the transformer model with a validation set of 0.1 or 10%. Before the dataset was split for training, a 1000 random holdout or testing set was taken for evaluatory reasons.

For **stage two**, The Bert model will be trained on a smaller dataset to check if having a slightly balanced class labels distribution in the dataset leads to a better performace. This was achieved by taking away 4500 examples from the positive class labels to balance the class distribution on the training and validation set. The same test train split was used as stage one.

1) Preparation of Dataset: For BERT to process the data, it needs to be converted into numerical representations, typically one would use word2vec or other algorithms for word embeddings, however for it to work with BERT, the sentences need to use BERT's tokenizer to encode it.

Before deciding the length of the sentences required to do these steps we found out the frequent amount of tokens needed for most of the sentences and based out the max length to that which happens to be about 256 tokens hence the max length was inputted in the encoding procedure.

Tokenization Steps

- Adding special tokens at the beginning and end of the sentences such as [CLS] and [SEP]
- Pad and truncate all the sentences to the total length of the [PAD]

- Explicitly differentiate real tokens from padding tokens with "attention masks" [MASK]
- Every sentence will be tokenized which each input id's and attention masks will be

This will be done for every sentence in the corpora

[CLS] and [SEP]

First the [CLS] token will be appended at the beginning of each sentence (Devin et. al, 2018). This is to signify that it is the beginning of a sentence whereas [SEP] is the end of the sentence. The [CLS] token is special because it represents the entire sentence or sequence into a "specific aggregate sequence representation for classification" (Ibid.). This basically means the CLS is a condensed token that has information about a specific sentence that can be used for classification purposes.

[PAD] All sequences passed into the BERT model are padded with extra tokens for it to match the total max length, so if a sentence is smaller than the max length specified that it will be padded with some extra tokens. This is one of the prerequisites for training with Bert

[MASK] Attention masks are basically denoting every token in the sequence as 1 and 0 indicating words are padded and not padded [4].

B. Experimentation Procedure

The entire training procedure is very simple. After the datset is preporcessed with BERT tokenizer it is then splitted using a training set of 90% and validation set of 10%. followed by converting the dataset into a data loader then finally training it. The training procedure was adapted from pytorch documentations [10]

1) System and Package Information: CUDA Version: 10.1 GPU: Tesla T4 IPython Version: 5.5.0 CPython Version: 3.6.9, Pytorch Version: 1.7.0+cu101 Transformers Version: 3.5.1 Numpy Version: 1.18.5 Pandas: 1.1.4

2) A Layer of Classifier: A layer of the classifier was built on top of pre-trained BERT model. This was adapted from Venlin [11]. Now this extra layer of a classifier uses the BERT model to extract the last hidden layer of the [CLS] token and put it through a single-hidden-layer feed-forward neural network as our classifier

The classifier itself will made through a class which will be extracted from the torch nn module. First, the input is fed in the Bert model, then dropout is added for regularization which then is followed by appending an output layer which will be a fully connected layer and then call a softmax from that output layer. The softmax is going to called at the end of the predict function. A dropout layer of 0.3 was specified for training the bigger dataset whereas, the layer was altered to 0.2 for training the smaller dataset.

3) Fine tuning parameters: The optimizer uses 5e-5 as it's learning rate and the epsilon value was 1e-8. The learning rates were recommended by the authors of the original BERT paper. Numbers of epochs used in this experiment is 4, the batch size was 32, also recommended by the authors (Devlin et. al, 2018).

4) Reproducibility and Loss Function: Seed value with 42 was added before training to all the libraries. For the loss function CrossEntropy was used from the pytorch library.

C. Evaluation Metric

All the metrics used for this re research paper uses sci-kit learns libraries to execute evaluation metrics.

ROC AUC: refers to Area under Receiver Operating Characteristics. Area under ROC curve essentially shows if the binary classifier can distinguish between positive to negative labels. Any binary classifier should strive to have a AUC from 0.5 to 1 which show it has a good capability of predicting class labels with high accuracy, whereas AUC equivalent to 0.5 or below is considered sub-par performance.

Accuracy: Accuracy essentially looks at all the true positive and negative predicted labels against all the predicted labels. The closer accuracy value is to 100% usually the better.

Precision: it looks at the proportions of positive labels that was actually correct. The values lies between 0 and 1. The closer it is to 1 the better.

$$Precision = \frac{TP}{TP + FP}$$

F1 score: also known as the harmonic mean of precision and recall, this is a better version of accuracy where it strikes a balance between precision and recall of a classifier.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

IV. Results

The results section will be in two parts as one part is going to be trained with a slightly larger unbalanced dataset and the other is going to be "balanced" and reduced dataset so that the positive and negative examples are the same proportions when training.

A. Stage 1

The section is going to show the training results on a holdout set of random 1000 examples where the positive examples were 703 and 297 negative examples to test on. Total examples the classifier is trained on is 8964 positive examples and 4332 negative examples. Before the test data were inputted they have all handled the same way as mentioned in data preprocessing step with data loaders

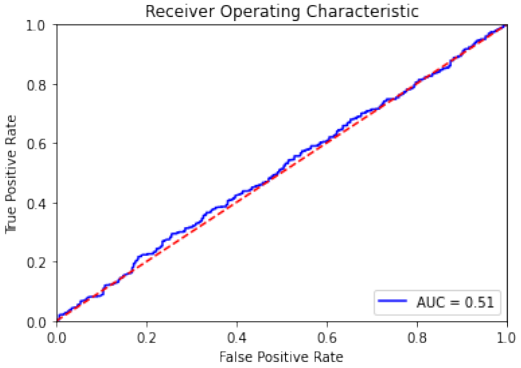

Figure 1 ROC AUC for stage 1

In figure 1 we can see we get an Area under ROC curve of 0.510 which is about 0.03 bigger than the model train on the smaller dataset in figure 2.

Epoch	Train Loss	Val Loss	Val Acc
1	0.144048	0.100067	96.43
2	0.059786	0.112545	96.58
3	0.021165	0.143412	97.02
4	0.008796	0.162449	96.80

Table 1 Training details on each epoch

As observed from Table 1, the validation loss at epoch 3 and 4 is greater than the training loss, although the validation accuracy peaks at 97% at epoch 3 but the conclusive validation accuracy is 96.70%, hence it can be derived that the classifier is overfitting on the validation set. The main point of this table is to observe the overfitting of the model training on each epoch rather than the actual values as they can slightly vary under reproduction.

Metrics	Results
AUC	0.510
Accuracy	58.80%
F1 Score	0.588790194609707
Precision Score	0.7081545064377682

Table 2 Results from experimental stage 1

B. Stage 2

This section of the experiment will be using a smaller dataset of 8219 examples where 4135 negative and 3994 positive examples. This is to see if the Bert model can still be trained to check sentiment polarity at a smaller dataset with a similar proportion of label distribution. Although it is to be noted that the dropout layer was set to 0.2 because of memory management otherwise the experiment simply fails to run.

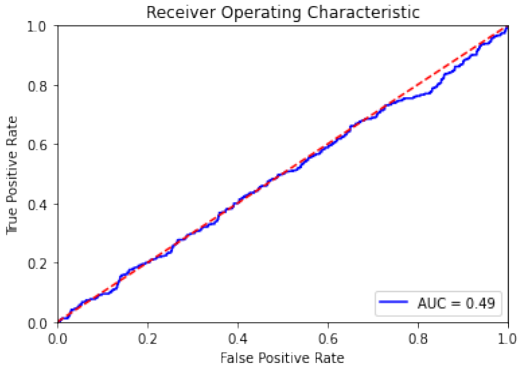

Figure 2 ROC AUC for stage 2

In figure 2 we can see we get an Area under ROC curve of 0.48 which is about 0.03 less than the model train on the bigger dataset in figure 1. But both Figure 1 and 2 look essentially they same. It is to be noted that Figure 2 rounds up 0.48 to 0.49.

Epoch	Train Loss	Val Loss	Val Acc
1	0.163518	0.130081	95.67
2	0.067859	0.112044	96.51
3	0.030878	0.155625	96.88
4	0.011544	0.180400	96.75

Table 3 Training details on each epoch

The same overfitting phenomena like Table 1 is observed at epoch 3 as the validation loss is greater than the training loss by a large margin. The main point of this table is to observe the overfitting of the model training on each epoch rather than the actual values as they can slightly vary under reproduction.

Metrics	Results
AUC	0.481
Accuracy	48.70%
F1 Score	0.4869769098047061
Precision Score	0.48688775163152864

Table 4 Results from experimental stage 2

V. Discussion

As observed in Table 1 and Table 3, there is a similar overfitting pattern on the training set. This could have been mitigated by increasing the validation set to 30-40% and observing how it would have affected the training and validation loss. Figure 1 and Figure 2 shows very similar ROC AUC value near 0.5 which means it is predicting randomly or poorly., although it is worth mentioning that stage 1 performed better in terms of accuracy by almost10% (Table 2 and 4) which means more training data might have helped with predictive performance. The results show that CNN performs significantly better for sentiment analysis task as it had an accuracy score of 95% (Related work, 2). This begs the validity of BERT being the state of the art when it comes to text classification.

However, it is worth denoting that there were difficulties when pre-training the KB-BERT [3]. Therefore, if more training data is fed for pre-training or fine-tuning, it could potentially enhance the performance of BERT, not only for binary classification but also multi. Thus, dismissing the promise of transformer-based sentiment analysis or other text classification performance doing better is too premature to decide. Additionally, one other possible way to increase accuracy for sentiment analysis is by increasing the dropout layer. It is also worth mentioning that increased amount of data might lead to better sentiment classification score [9]

VI. Conclusion

Conclusively, The results show fine-tuning with 14,000 data points gives an accuracy of 60% which means more data is needed. About 5000 to 6000 extra data points might be needed as per paper from LTH [7]. On the other hand, if the KB BERT model is pre-trained with even more training data, that could also reduce the amount of data needed to fine-tune. This is further proven from stage 2 experiment as the accuracy was down by 10% showing less data or balanced class labels of data does not increase accuracy. One could even experiment with changing some hyper parameters, such

as increasing the dropout layer to 0.5 or decreasing it to 0.1 and maybe decreasing the batch size to 16 as recommended by the authors of BERT. Future research could be done with multi labelled classification, but feeding more data into the KB BERT is highly recommended for fine tuning. Convolutional neural network are shown to be more effective at classsiyfing sentiment with the given amount of data. However, there is still hope for the swedish BERT model as, if KB BERT is trained with more data this could allow BERT to classify multi labeled sentiments with very high accuracy.

Acknowledgment

The author would like to thank Kristoffer for providing training data set for strict academic reasons [9]

Special thanks to Nyamgarig and Jimmy who proofread this research paper [6]

References

References

[1] Jay Alammar. *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. URL: http://jalammar.github.io/illustrated-bert/.

[2] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[3] Martin Malmsten, Love Börjeson, and Chris Haffenden. "Playing with Words at the National Library of Sweden–Making a Swedish BERT". In: *arXiv preprint arXiv:2007.01658* (2020).

[4] Chris McCormick. *BERT Research - Ep. 3 - Fine Tuning - p.1*. Dec. 2019. URL: https://www.youtube.com/watch?v=x66kkDnbzi4.

[5] Derek Miller. "Leveraging BERT for extractive text summarization on lectures". In: *arXiv preprint arXiv:1906.04165* (2019).

[6] Nyamgarig Naranbaatar. *Nyamgarig Enzo Naranbaatar*. URL: https://www.linkedin.com/in/enzittonn/.

[7] Fredrik Olsson and Gustav Handmark. "Sentiment Analysis and Aspect Extraction for Business Intelligence". In: *LU-CS-EX* (2020).

[8] Matthew E Peters et al. "Dissecting contextual word embeddings: Architecture and representation". In: *arXiv preprint arXiv:1808.08949* (2018).

[9] Kristoffer Svensson. *Sentiment analysis with convolutional neural networks: classifying sentiment in Swedish reviews*. 2017.

[10] *Training a Classifier*. URL: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.

[11] Venelin Valkov. *Sentiment Analysis with BERT and Transformers by Hugging Face using PyTorch and Python*. Apr. 2020. URL: https://curiousily.com/posts/sentiment-analysis-with-bert-and-hugging-face-using-pytorch-and-python/.

[12] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017), pp. 5998–6008.

[13] Quanbin Wang and Ying Tan. "Grammatical Error Detection with Self Attention by Pairwise Training". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–7.

[14] Lilian Weng. *Attention? Attention!* June 2018. URL: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html.