EEE 486/586 Statistical Foundations of Natural Language Processing Assignment 2

Mehmet Rıfat Özkurt, *B.Sc., 22003187, Bilkent University Electrical & Electronics Engineering Department*

***Abstract*—In this study, a pre-trained BERT language model was fine-tuned for the classification of English sentences according to their grammatical correctness. An initial pre-trained BERT model ('bert-base-uncased') trained as a masked language model was chosen to be trained with the labeled CoLA dataset from GLUE Benchmark Datasets. The evaluation of the results were done through Matthew's Correlation Coefficient and evaluation loss. For the classification, two methods were employed: the traditional ['CLS'] token method which handles a fixed-length ['CLS'] vector that summarizes the contextual information in a sequence, and a Maximum Pooling method which outputs a fixed-size token generated from the maximum values in the hidden layers of the pretrained model. The trial parameters were optimized using Optuna. The first method was run on the Huggingface API using BertForSequenceClassification model, which yielded a best run with MCC = 0.58 and loss = 0.44. The second method was run on a local machine using PyTorch in an Anaconda environment, by defining a subclass to the BertForSequenceClassification model with the second method, which yielded a best run with MCC = 0.39 and loss = 0.40. All in all, the second method performed much poorly than the first, both due to a significant decrease at MCC in local runs, and presumably due to the unsuitable model structure. Overall, smaller batch sizes and learning rates are found to be the safer parameters.**

***Index Terms*—Natural Language Processing, BERT, sentence classification, CoLA, Maximum Pooling, Huggingface, CLS**

I. Introduction

THIS paper is aimed at discussing the methodologies and the results of two fine-tuning experiments performed on the Bidirectional Encoder Representations from Transformers (BERT) language model, using the Corpus of Linguistic Acceptability (CoLA) dataset from GLUE Benchmark Datasets. The selected BERT model focuses on sentence classification, which outputs whether an English sentence is grammatically correct or not, after the fine-tuning with the CoLA dataset. Two different fine-tuning methods were tested on the BERT model. The first of these methods employs the ['CLS'] token, where the encoding of the input sequence generates an output for the ['CLS'] token that summarizes the input sequence in a vector. This token is then fed to a final classification layer, which is a fully connected layer and can be trained alongside the rest of the model to generate an output classification. The second method does not involve the ['CLS'] token and uses the hidden states of all the tokens in the input sequence, applies max pooling to obtain the maximum hidden state values across all the tokens and feeds these to the final classification layer. The first method is traditionally called the traditional ['CLS'] token approach and the second method is known as max pooling approach. Note that the second method is an arbitrary and experimental proposal and does not claim to output more accurate results than the first method in this experiment. This paper will introduce the main concepts about BERT model and fine-tuning, before presenting the architecture of the second model and presenting the results.

A. BERT Model and Transformers

Transformers are a type of neural network architecture that are one of the most commonly utilized models for NLP purposes. They offer newer mechanisms compared to RNNs and FFNs, which are self-attention and positional encoding [1]. Self-attention is a mechanism that aids the extraction of context from a sequence of words by taking into account the relevance of a word with the other preceding words in the sequence and computing a weighted sum of the input sequence at each position, depending on these relevancies. The relevance in this context is represented by a pairwise score calculation for each pair of words that precede the selected word. This score is then normalized using softmax to create relevant weights and the sum is calculated for one layer. One other advantage of transformers is that these attention layers can be stacked, allowing for multiple levels of abstraction [2]. A typical transformer consists of these attention layers and feed forward layers. The training is done by error calculation and gradient descent methods, which automatically update the weights within the model. Fig 1 represents a common transformer.

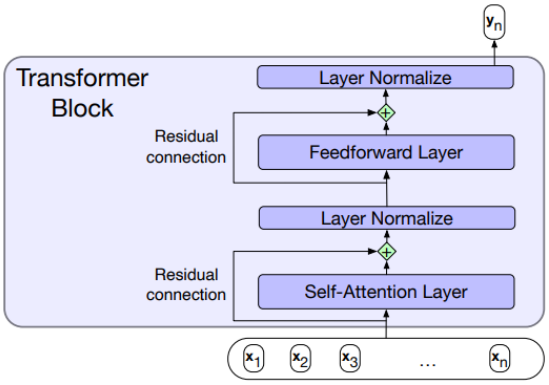

Fig. 1 The composition of a common tranformer block, with attention layers, normalization layers and feedforward layers [1].

BERT is a pre-trained language model that takes advantage of the attention mechanism and a transformer-based architecture to train on the conceptual meaning of words and sentences. However, the attention (self-attention) mechanism in BERT is not the same as other common models. In traditional transformer-based models, the selected token can be related to preceding tokens in a sequence but not the trailing tokens. This prevents the algorithm to fully understand the meaning of a word in a sentence and catch all the nuances. BERT's self-attention mechanism (bidirectional encoding) can calculate the relationship between the current token and all of the other tokens in a sequence, giving it a broader access and enabling it to better summarize the meaning of a token in the given context [2]. BERT is trained through unsupervised learning on a large corpus, with the objective of prediction of the next word. This enables the contextualization of word embedding ins a general context. The bidirectional encoding mechanism of BERT, however, makes it impractical to predict the next word in a given sequence, since the algorithm already has access to the next word and can output it with perfect accuracy. This is tackled by employing a masking method, namely Masked Language Modeling, that masks a certain percentage of random tokens in a sequence and trains the model to predict the masked token. MLM trains BERT to encode rich contextual information for each token, since it has to consider the position of the masked word in the sentence and predict the word with respect to the surrounding words.

B. Fine-tuning BERT Models

Since BERT is trained on very large corpora in an unsupervised manner, its rich general contextual understanding of word embeddings can be used in general NLP tasks. However, BERT can also be fine-tuned for specific NLP tasks to improve its performance. Fine-tuning BERT requires the pre-trained BERT model to be trained again on an application-specific corpus, which is usually smaller than the original corpus BERT was trained on. This retraining of the model allows for application-specific context to be embedded into the words, usually by changing some of the weights of the classifier head [1]. For instance if a word has an uncommon meaning in a specific area, BERT will be able to link that meaning to the word after this retraining. This is more commonly known as transfer learning, where the knowledge in a general area is linked to a more specific one. In this experiment, the specific application was to classify a sentence as grammatically correct or not. For this purpose, a pre-trained model BERT model "bert-base-uncased" is retrained on the GLUE benchmark dataset CoLA. This dataset includes a large corpus of individual sentences that are labelled as grammatically correct or incorrect, called sequence classification. In BERT, the most common method to do this is to produce the aforementioned ['CLS'] token and concatenate it with the sequence. This token will inherit all the contextual information of the sentence into one vector, which will later be outputted into a classifier head to determine a final output. Part 1 of this assignment (not mentioned in Architectures) uses this technique to classify sentences, where Part 2 uses another technique to exclude the ['CLS'] token from the classifier head. Fig.2 represents the general sequence classification model.

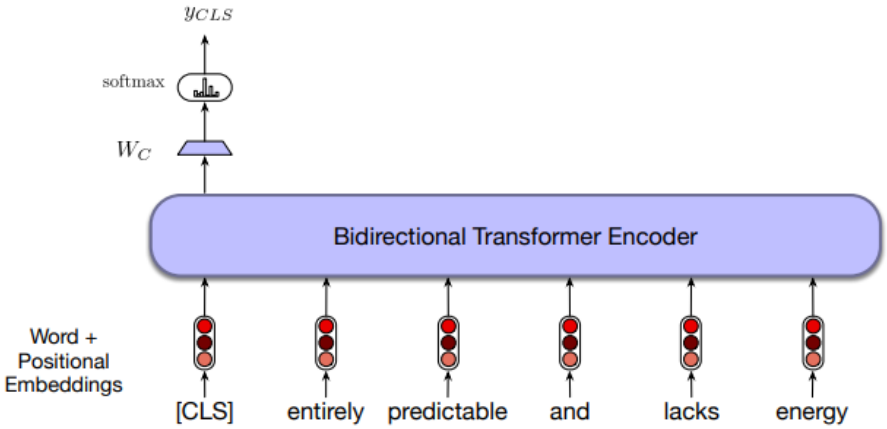

Fig. 2 A bidirectional transformer encoder model, similar to BERT, for sequence classification. The concatenation of the ['CLS'] token enables the model to represent the information of the sequence in one classifier, which is outputted to a classifier head [1].

II. ARCHITECTURES

As mentioned before, Part 1 of this assignment uses the traditional model with [‘CLS’] classifier vector for ‘sentence embeddings’. In Part 2, this approach is forgone and a common attention pooling method is employed, called Max Pooling. This method extracts a fixed-size representation of an input sequence, similar to the CLS method. In this case, this output vector is obtained by taking the maximum hidden state values from all the tokens in the input sequence. In this experiment, all the hidden states are obtained from the pre-trained BERT model, which are the outputs of the last layer of BERT. Then, the maximum values across all these tokens is collected into a single vector and fed into the next fully connected network, which is the classifier head. In this method, the maximum value vector replaces the [‘CLS’] token, which is an arbitrary choice. This classifier head then a softmax function to generate a probability function for the classification. The output of the whole model is this probability function. Fig.3 shows a simplified diagram of the process. A mathematical representation is below, which can be understood with reference to Fig.3. For this mathematical representation, the pre-trained BERT model is shown as BERT() function, whose outputs are the last hidden layer and the classifier head outputs CH.

$$y_n = BERT(x_n) \tag{1}$$

$$y_{\max\_pooled}[i,j,k,\dots] = \max(y_1[ii,j,k,\dots], y_2[i,j,k,\dots], \dots, y_n[ii,j,k,\dots]) \tag{2}$$

$$y_{\text{prediction}} = softmax(CH(y_{\max\_pooled})) \tag{3}$$

To really achieve this on PythonThe output of the model is a prediction, which indicates whether a given sentence is grammatically correct or not. To achieve this, the pre-trained BERT model is enhanced with fine-tuning, as expressed in the introduction. Labeled sequences from the CoLA dataset are fed into the model for training. The output $y_{prediction}$ and the actual training label are then given to an error function (crossentropy loss function) to determine the error of the function. This error vector then propagates back into the weights of the classifier head to adjust to the new dataset. To carry out this training, a few hyperparameters are needed, such as the batch size, maximum length, learning rate, number of epochs and dropout. Batch size is the number of samples that are fed in one training, where a higher batch size may cause memory overload problems and a smaller batch size will increase the training time. Maximum length refers to the number of tokens in the input data and is not very important for our purposes. Learning rate refers to the back propagation and gradient descent methods and specifies the size of the steps in these methods. A larger step means that with each iteration, the weights change more drastically, with the risk of overstepping a minimum optimal value. A smaller step size can ensure that no minimum is overlooked at the expense of very slow training. The number of epochs refers to how many times the model will be trained with the same parameters, and usually more epochs can improve the accuracy of the model. The dropout is the chance of a random node being set to zero, to prevent the network from being too dependent on specific nodes, used to regulate the neural network. An open-source library called Optuna will train the model with different hyperparameters and evaluate the runs according to the loss and the Matthew Correlation Coefficient, calculated with reference to the labeled data and the output. The best run will be output by Optuna, and the model will be re-trained using those hyperparameters. The tuning parameters list is given below:

param_dict = {"lr": [1e-6, 1e-5], "n_epochs": [1, 2], "max_length": [16, 32], "batch_size": [16, 32, 64]}

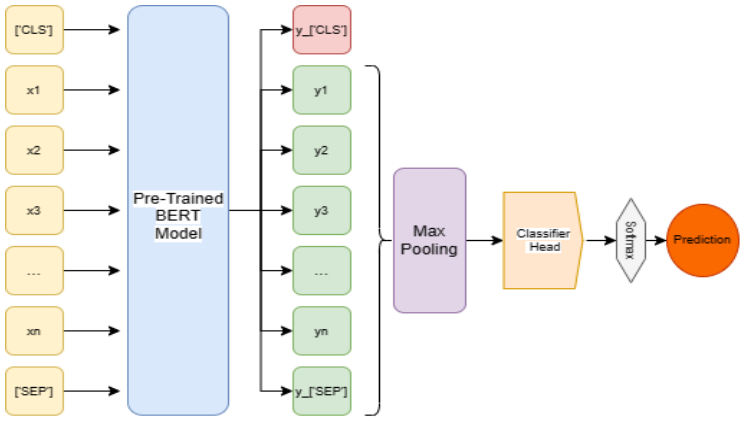

Fig. 3 An overall representation of a Max Pooling method on a pre-trained BERT model, where the ['CLS'] token is disregarded and maximum values of the hidden layers are collected into one vector, fed to the classifier head and softmax to generate a final prediction.

In the proceeding part, the results of these hyperparameter tests will be given with the chosen best runs. The evaluation metric for the runs is MCC and the loss coefficient.

III. Results

The first part of the experiment was run on a Huggingface training notebook on Google Colab. This part uses the API of Hugging face and the whole trained model can be reached from this hyperlink. The choosing of the hyperparameters is done both manually and by using Optuna. For the second part, the BERT model was imported to a local Anaconda environment and PyTorch libraries were used in Python for vector operations. Cuda is also used for GPU acceleration on the local machine. The results of these runs will be presented in the proceeding sections.

A. Results of Part 1: ['CLS'] Method

In this part, the API of Huggingface was used to call and use BERT model, alongside CoLA dataset. BertForSequenceClassification is used as the chosen model, which has an untrained classifier head, suitable for fine-tuning. The hyperparameter range is given to the Optuna library, which performs the training runs. The model was trained for 6 times for a total of 3 trials. Number of epochs, learning rate, maximum length and batch size were changed, dropout was constant. Below are the raw results.

TABLE I
TRIAL RUN RESULTS OF THE ['CLS'] TOKEN METHOD FOR DIFFERENT HYPERPARAMETERS (BEST RUN GREEN)
EACH TRIAL HAS ITS OWN EPOCH SEQUENCE DEPENDING ON THE NUMBER OF EPOCHS WANTED. THE MATTHEW CORRELATION COEFFICIENT AND THE LOSS ARE (BUT ARE NOT LIMITED TO) THE SUCCESS METRICS OF THIS MODEL

Trial #	Epochs	Learning Rate	Batch Size	MCC	Loss
0	1	2.03e-06	64	0	0.62
1	1	8.30e-06	16	0.4895	0.48
1	2	8.30e-06	16	0.5806	0.44
1	3	8.30e-06	16	0.5704	0.53

The table above shows the two trials of training on Huggingface API. The best run is evaluated with an MCC of 0.58 and a loss of 0.44. As can be observed from the table, MCC equals zero in the initial training. This may be due to two factors: BertFor SequenceClassification comes with an untrained classifier head and the first evaluation reflects this, or a very low learning rate and the other input parameters coincide at an MCC of 0. Judging by the high loss, there may actually be a MCC that is very close to 0, which was rounded. As the learning rate increases in the second trial, MCC increases with a large step. Considering that MCC ranges from -1 to 1, the maximum MCC suggests that there is some level of promising correlation. In fact, when Huggingface API is used to input random sentences, the model labels random arbitrary sentences correctly. One interesting phenomenon is the fall in the MCC and the rise in the loss as epoch changes from 2 to 3. Normally, increasing epochs should increase the accuracy since the model is trained for longer. This negative effect is probably due to the very large step size, where the back propagation in epoch 2 changes the weights by too much and the model oversteps the optimal point in the gradient descent. This can perhaps be solved with a smaller step size. The best result is obtained with the lowest batch size, which may underline some hardware constraints.

B. Results of Part 2: Max Pooling Method

To implement the Maximum Pooling Method, BertForSequenceClassification object was imported to python and a subclass to that was defined. The forward function in this subclass was changed to pool the maximum values of the hidden layers, rather than directly outputting the [‘CLS’] token. Below are the raw results for the Maximum Pooling method.

TABLE I
EACH TRIAL HAS ITS OWN EPOCH SEQUENCE DEPENDING ON THE NUMBER OF EPOCHS WANTED. THE MATTHEW CORRELATION COEFFICIENT AND THE LOSS ARE (BUT ARE NOT LIMITED TO) THE SUCCESS METRICS OF THIS MODEL

Trial #	Epochs	Dropout	Learning Rate	Maximum Length	Batch Size	MCC	Loss
0	1	0.2	9.75e-06	16	32	0.2942	0.54
1	1	0.2	4.23e-06	32	32	0.0401	0.58
2	1	0.2	6.95e-06	32	16	0.4011	0.52
2	2	0.2	6.95e-06	32	16	0.3913	0.40
2	1	0.2	6.95e-06	32	16	0.4100	0.52
2	2	0.2	6.95e-06	32	16	0.3831	0.39

Based on the table above, it can be observed that the first and the second trials performed much more poorly than the third trial (Trial #2). This may be because of the overshooting/undershooting of the learning rate, since the MCC coefficient drastically increases with a learning rate in between in the third trial. Within the third trial, it is seen that as epochs progress, the results tend to get more accurate, as the model is more trained. However, it is seen that a slight decrease in the MCC is observed in the second epochs, where a larger decrease is observed in the loss, for both runs. The decrease in the MCC may be explained as an overstepping, or a sacrifice to decrease the loss output. Compared to the previous model (Part 1), a significant drop in the MCC values is observed. This may be due to the incompetence of the newly-proposed model for this task, or a constraint introduced by running the trainings on a local machine with PyTorch. Besides the incompetence of the model, free trial local runs of the CLS method showed that the same method performs much poorly on the local machine than on Huggingface, showing that some hardware issues or problems with PyTorch are present. The best result is obtained with the lowest batch size, which strengthens the hardware -issues claim. The MCC value is 0.39 for the best run, and 0.41 at maximum. In fact, the same parameters run twice outputted similar but different results, depending on the many random factors, one of which is dropout.

IV. DISCUSSIONS & CONCLUSIONS

The aim of this assignment was to fine-tune a pre-trained BERT language model for sentence classification using a database from the GLUE Benchmark datasets, CoLA. Both CLS method and maximum pooling method were tried. The scripts were run locally and on Huggingface API. The results are presented and evaluated.

It is seen that the CLS model has significantly higher MCC values than Maximum Pooling. Part of this may be because the first algorithm is better suited for this task or better optimized. Probably, choosing the maximum values from each word embedding without the [‘CLS’] token removes a lot of contextual information from the output sequence. However, the CLS model run on the local machine still performs poorly compared to the one run on Huggingface, as was observed at an unrecorded trial. This problem may have arised from the hardware incompatibilities, an internal feature of PyTorch or another different property of the Huggingface model. Perhaps with a different model, local runs may yield better runs than Huggingface.

It is seen that epoch number is no direct method of increasing accuracy, and in some cases it yielded a worse run than the previous. This may be due to a large step size, where the new back propagation changes the weights by too much. Nevertheless, in some trials the MCC values seemed to have dropped very slightly but the loss decreased significantly as epochs progressed. Perhaps a tradeoff is made between the two output parameters. Overall, very high learning rates never yielded the best runs.

It is seen that the best runs are observed on the smallest batch sizes. This may be an indicator that the hardware that runs the trainings requires smaller loads and performs better under these conditions.

All in all, this assignment was focused on fine-tuning a pre-trained model, and introducing common NLP methods, practices, libraries and programs. Fine-tuning a BERT model and using the Huggingface API, Huggingface version control, creating an environment locally, taking advantage of GPU acceleration, using PyTorch libraries, using Optuna for hyperparameters and defining a subclass for a model were covered. I learned most of the skills mentioned above from scratch, with the help of the tutorials.

REFERENCES

[1] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Stanford, CA, USA: Thomson Higher Education, 2020. [Online]. Available: https://web.stanford.edu/jurafsky/slp3/. [Accessed: May 4, 2023].

[2] H. Face, ‘The Hugging Face Course, 2022’, 2022. [Online]. Available: https://huggingface.co/course.