# Natural Language Processing
# Final Project Report

**Topic: GLUE Tasks**

Dec 5, 2021

Group 3:
Abdulaziz Gebril
Yuan Meng
Yuchen Ma

## Table of Contents

## 1. Introduction.

Our project aims to apply the transformers models to complete the General Language Understanding Evaluation (GLUE) benchmark [1] tasks, a collection of natural language understanding systems for training, evaluating, and analyzing, containing Single-Sentence Similarity and Paraphrase Tasks, and Inference Tasks. It is a widely used benchmark, so we can solve the same problem with different teams worldwide to get more inspiration.

GLUE consists of nine language understanding tasks built on established existing datasets, introduced in the next section.  And the transformers [2] we chose for these tasks are the BERT and the RoBERTa models, which are discussed in the third section. Then, for the fourth section of the report, we explained the experimental setups and talked about the hyper-parameters for fine-tuning models in the fifth section. After that, we discussed our results in the sixth part of our report, and then the summary and conclusion are as follows. In the end, we listed our references.

## 2. Description of the data set.

The GLUE Benchmark datasets include The Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2018) consists of 10657 sentences from 23 linguistics publications; The Stanford Sentiment Treebank(SST-2) (Socher et al., 2013) consists of 9645 sentences; Microsoft Research Paraphrase Corpus for Similarity and Paraphrase Task (MRPC) (Dolan & Brockett, 2005) containing 5800 pairs of sentences; Semantic Textual Similarity Benchmark (STS-B)(Cer et al., 2017) comprising 8628 sentence pairs; Quora Question Pairs (QQP) collecting question pairs from the website Quora; MultiNLI Matched (MNLI)(Williams et al., 2018) consists of 433 thousand sentence pairs; Question NLI (QNLI) (Rajpurkar et al. 2016) containing more than 100000 question-answer pairs from more than 500 articles; Recognizing Textual Entailment (RTE) combining the data from RTE1 (Dagan et al., 2006), RTE2 (Bar Haim et al., 2006), RTE3 (Giampiccolo et al., 2007), and RTE5 (Bentivogli et al., 2009); and Winograd NLI (WNLI) (Levesque et al., 2011) collecting 150 Winograd schemas.

The table below shows the structure of the datasets.

| Datasets | Column 1 | Column 2 |
|---|---|---|
| CoLA | Sentence | None |
| SST-2 | Sentence | None |
| MRPC | Sentence1 | Sentence 2 |
| STS-B | Sentence1 | Sentence 2 |

| | | |
|---|---|---|
| QQP | Question 1 | Question 2 |
| MNLI | Premise | Hypothesis |
| QNLI | Question | Sentence |
| RTE | Sentence1 | Sentence 2 |
| WNLI | Sentence1 | Sentence 2 |

## 3. Description of the NLP model.

**BERT Architecture**

Bidirectional Encoder Representations from Transformers is a language representation model. It was introduced by Google in 2018 (https://arxiv.org/pdf/1810.04805.pdf) Unlike other language representation models such as ELMO or OpenAI GPT, BERT is pretrained on both left and right context. The BERT model uses a masked language model pre-training objective. This is achieved by randomly masking some of the tokens from the input, and the objective is to train the masked token based on the bidirectional context. The model also uses the next sentence prediction task that predicts text-pair representation.

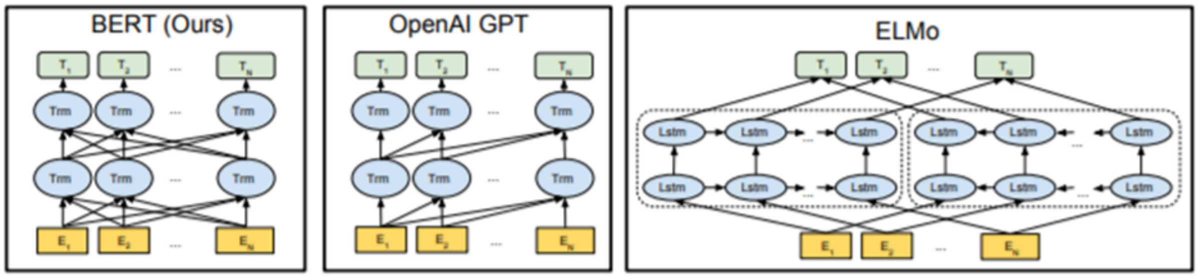


Figure 1. Comparison between BERT, OpenAI GPT and ELMO

https://arxiv.org/pdf/1810.04805.pdf

The BERT model has different sizes, BERT base model includes 12 Transformers blocks, a hidden size of 768 and 12 self-attention heads. The BERT large model includes 24 Transformers blocks, a hidden size of 1024 and 16 self-attention heads. BERT model can take an input representation that is able to represent both a single and a pair of sentences. This enables the model to handle a variety of down-stream tasks. BERT model uses WordPiece embeddings with a 30,000 token vocabulary. As seen in figure 2, each token embedding is a result of the addition of the segment and position embeddings.

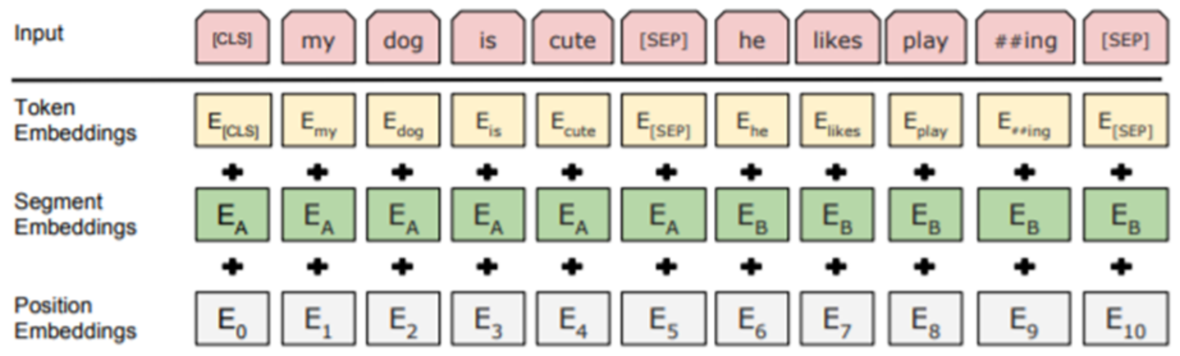

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ $E_{my}$ $E_{dog}$ $E_{is}$ $E_{cute}$ $E_{[SEP]}$ $E_{he}$ $E_{likes}$ $E_{play}$ $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
$E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_B$ $E_B$ $E_B$ $E_B$ $E_B$
Position Embeddings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

Figure 2. Bert Input Representation

https://arxiv.org/pdf/1810.04805.pdf

The following tokens are considered as special tokens used for various purposes.

· [CLS], The first token of every sequence is always a special classification token. Also, the final hidden state of this token is used for classification tasks.

· [SEP], token to separate sequence inputs generated by combining sentence pairs.

· [MASK], token used during MLM procedure to replace the masked word.

The first objective during training the model is the Masked language model (MLM) involves masking 15% of all WordPiece tokens in each sequence at random. 80% of the time, the [MASK] token replaces the word, 10% the token is replaced with a random token and 10% the token is left as is.

The second objective is the Next sentence prediction (NSP). Language modeling does not capture the relationship between two sentences, which is imperative for some Natural Language Processing tasks such as Question Answering and Natural Language Inference. To achieve this, the model is pre-trained on a next sentence prediction. This is done by providing different examples where 50% of the time, the two sentences entail each other.

The data used to pre-train the BERT model is the BookCorups which contains 800 Million words and English Wikipedia ( only text) which contains 2,500 Million words.

## RoBERTa Architecture

Roberta stands for Robustly Optimized BERT Pretraining Approach. From its name , Roberta was introduced by Facebook and University of Washington (https://arxiv.org/pdf/1907.11692.pdf) as a replication study of BERT pre-training, which includes a careful evaluation of the effects of hyperparameter tuning and training set size. The paper discussed that the BERT model was significantly undertrained. The RoBERTa has the same architecture as BERT. The main differences between BERT and RoBERTa are as follows;

· Training RoBERTa was longer, with bigger batches over more data.

· Removing the next Sentence objective.

· Training on longer sequences.

· Using Dynamic Masking in the masking language model objective.

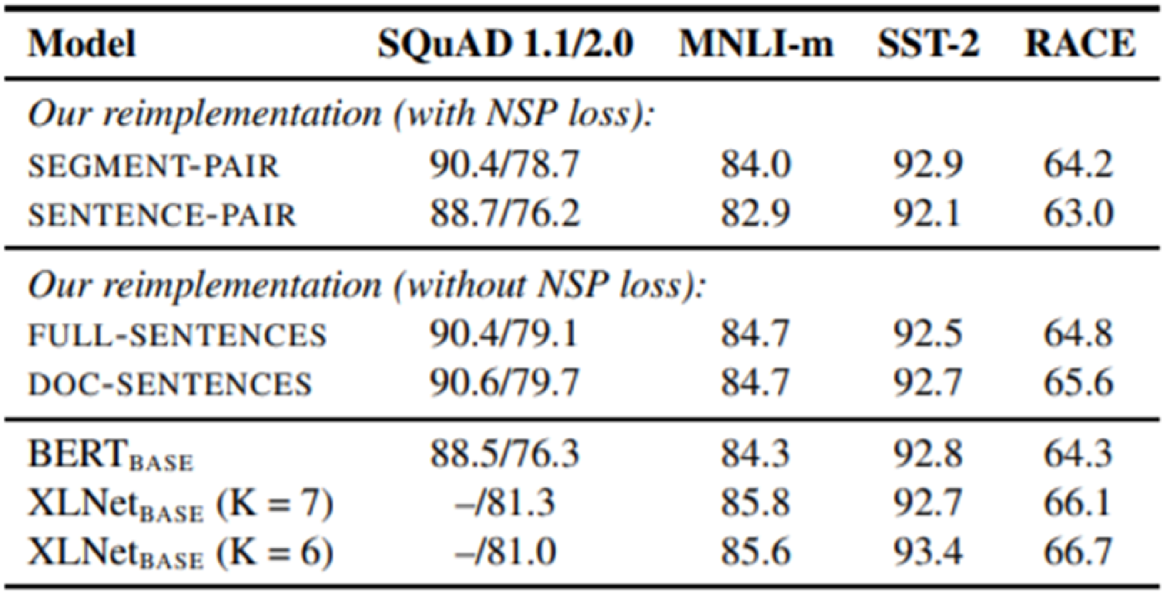
The paper included testing the necessity of using the Next Sentence objective. This was achieved by testing the using various training formats. Figure 3 shows results of pre-training using these different formats.

· Segment-Pair with NSP loss, this is the same next sentence objective used in BERT pre-training.

· Sentence-Pair with NSP loss, each input contains a pair of natural sentences sampled from same or separate documents.

· Full-Sentences without NSP Loss, each input is packed with full sentences from one or more documents.

· Doc-Sentences without NSP, each input are also packed with full sentences but from same document.

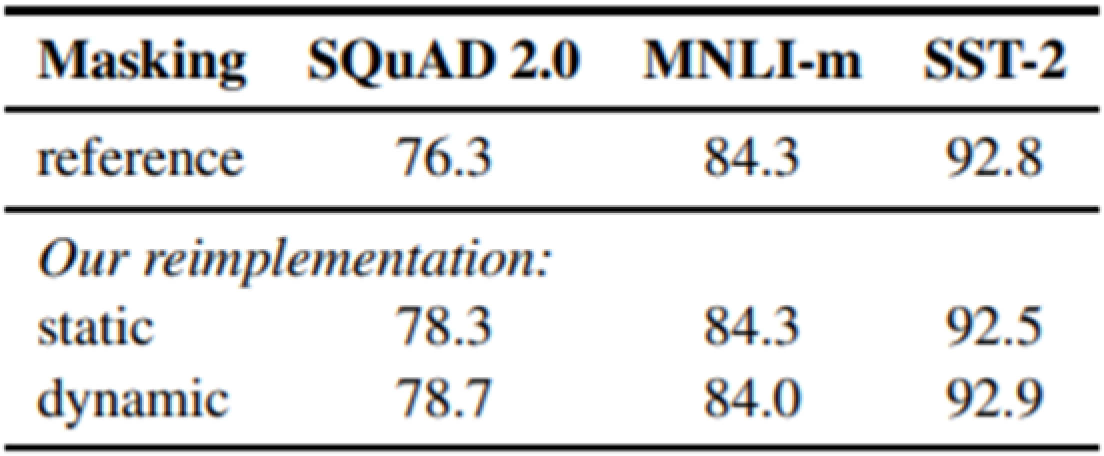


| Model | SQuAD 1.1/2.0 | MNLI-m | SST-2 | RACE |
|---|---|---|---|---|
| *Our reimplementation (with NSP loss):* | | | | |
| SEGMENT-PAIR | 90.4/78.7 | 84.0 | 92.9 | 64.2 |
| SENTENCE-PAIR | 88.7/76.2 | 82.9 | 92.1 | 63.0 |
| *Our reimplementation (without NSP loss):* | | | | |
| FULL-SENTENCES | 90.4/79.1 | 84.7 | 92.5 | 64.8 |
| DOC-SENTENCES | 90.6/79.7 | 84.7 | 92.7 | 65.6 |
| $\text{BERT}_{\text{BASE}}$ | 88.5/76.3 | 84.3 | 92.8 | 64.3 |
| $\text{XLNet}_{\text{BASE}}$ (K = 7) | –/81.3 | 85.8 | 92.7 | 66.1 |
| $\text{XLNet}_{\text{BASE}}$ (K = 6) | –/81.0 | 85.6 | 93.4 | 66.7 |

Figure 3. Results of different training formats on different downstream tasks

https://arxiv.org/pdf/1907.11692.pdf

Another difference discussed was the usage of dynamic masked language models. The BERT model was pre-trained using static masking, where the masking was randomly done during preprocessing. This results in training sequences with the same masks for every training epoch. RoBERTa was trained using dynamic masking, where the training data was duplicated 10 times so that each sequence is masked in 10 different ways over the training epochs. Figure 4 illustrated the difference in performance using each masked language model.



| Masking | SQuAD 2.0 | MNLI-m | SST-2 |
|---|---|---|---|
| reference | 76.3 | 84.3 | 92.8 |
| *Our reimplementation:* | | | |
| static | 78.3 | 84.3 | 92.5 |
| dynamic | 78.7 | 84.0 | 92.9 |

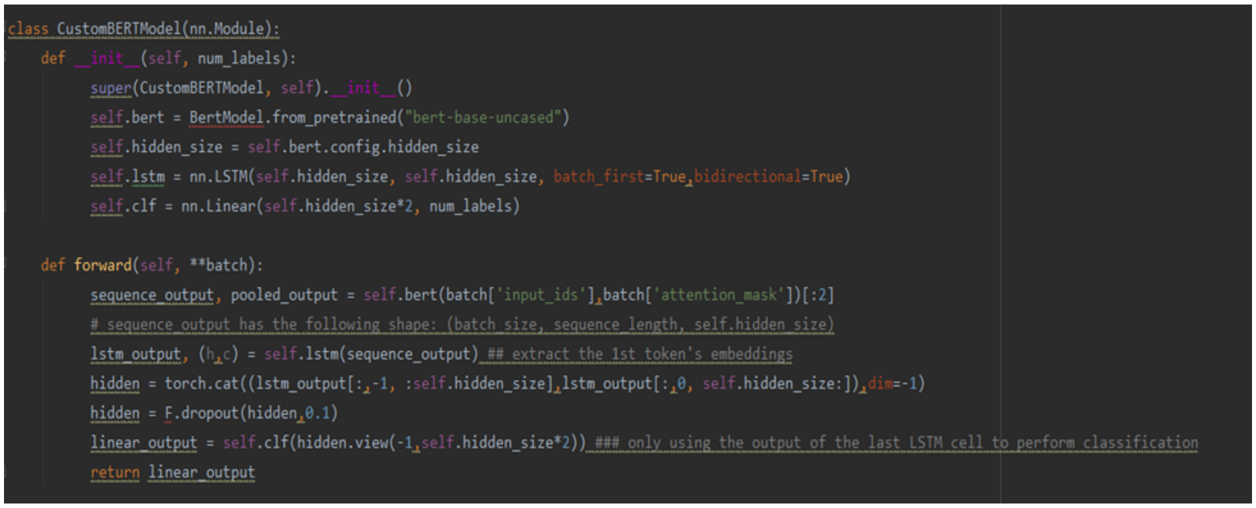
Figure 4. Dynamic vs Static masked language model

https://arxiv.org/pdf/1907.11692.pdf

RoBERTa model was trained using more data, The model was pre-trained on Bookcorpus and English Wikipedia. In addition to CC-news which includes the English portion of CommonCrawl news dataset. Also, OpenWebText and Stories which is a subset of the CommonCrawl news dataset.

## Custom BERT and Custom Roberta

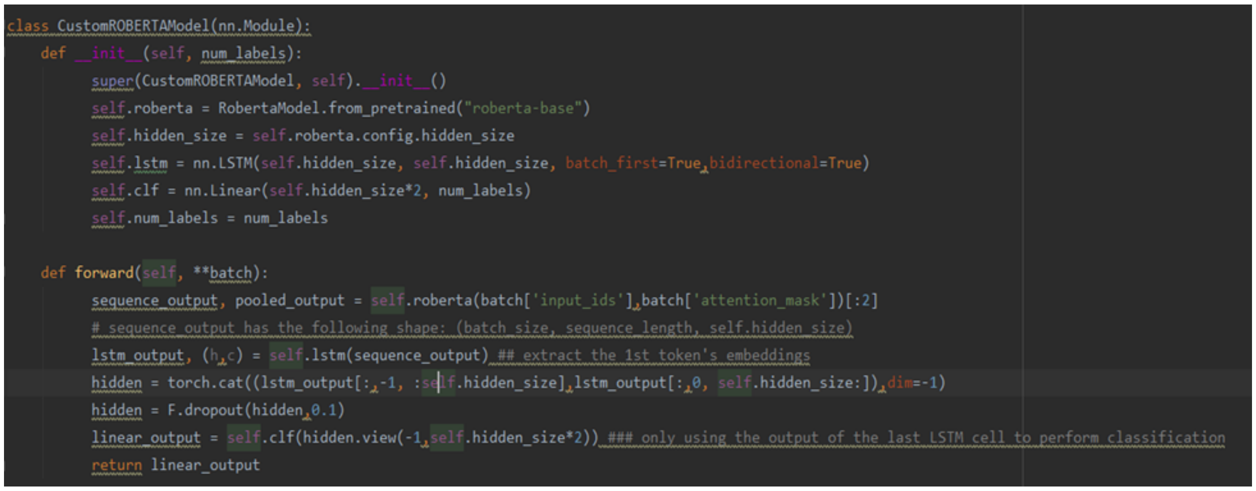
In this project, we used both pretrained BERT and RoBERTa models and added two additional layers. One bidirectional LSTM layer and a layer for model output for different GLUE natural language processing tasks. Figure 5 and Figure 6  illustrate code for the custom BERT and RoBERTA models used to train GLUE datasets.



```python
class CustomBERTModel(nn.Module):
    def __init__(self, num_labels):
        super(CustomBERTModel, self).__init__()
        self.bert = BertModel.from_pretrained("bert-base-uncased")
        self.hidden_size = self.bert.config.hidden_size
        self.lstm = nn.LSTM(self.hidden_size, self.hidden_size, batch_first=True,bidirectional=True)
        self.clf = nn.Linear(self.hidden_size*2, num_labels)

    def forward(self, **batch):
        sequence_output, pooled_output = self.bert(batch['input_ids'],batch['attention_mask'])[:2]
        # sequence_output has the following shape: (batch_size, sequence_length, self.hidden_size)
        lstm_output, (h,c) = self.lstm(sequence_output) ## extract the 1st token's embeddings
        hidden = torch.cat((lstm_output[:,-1, :self.hidden_size],lstm_output[:,0, self.hidden_size:]),dim=-1)
        hidden = F.dropout(hidden,0.1)
        linear_output = self.clf(hidden.view(-1,self.hidden_size*2)) ### only using the output of the last LSTM cell to perform classification
        return linear_output
```

Figure 5. Custom BERT model



```python
class CustomROBERTAModel(nn.Module):
    def __init__(self, num_labels):
        super(CustomROBERTAModel, self).__init__()
        self.roberta = RobertaModel.from_pretrained("roberta-base")
        self.hidden_size = self.roberta.config.hidden_size
        self.lstm = nn.LSTM(self.hidden_size, self.hidden_size, batch_first=True,bidirectional=True)
        self.clf = nn.Linear(self.hidden_size*2, num_labels)
        self.num_labels = num_labels

    def forward(self, **batch):
        sequence_output, pooled_output = self.roberta(batch['input_ids'],batch['attention_mask'])[:2]
        # sequence_output has the following shape: (batch_size, sequence_length, self.hidden_size)
        lstm_output, (h,c) = self.lstm(sequence_output) ## extract the 1st token's embeddings
        hidden = torch.cat((lstm_output[:,-1, :self.hidden_size],lstm_output[:,0, self.hidden_size:]),dim=-1)
        hidden = F.dropout(hidden,0.1)
        linear_output = self.clf(hidden.view(-1,self.hidden_size*2)) ### only using the output of the last LSTM cell to perform classification
        return linear_output
```

Figure 6. Custom RoBERTa model

## 4. Experimental setup.

First, for the data preprocessing part, we loaded the datasets from the load_dataset method of the datasets library, and tokenized them using the AutoTokenizer method from the transformers library. The tokenizing method depended on the model we chose. For instance, if we chose to train on the BERT model, we would define the tokenizer as AutoTokenizer.from_pretrained("bert-base-uncased", use_fast=True). We then padded the data and deleted the original columns, so the data only remained 'attention_mask', 'input_ids', 'labels', and 'token_type_ids' columns. Next, we loaded them to the training and validation data loader separately with defined batch size.

Second, we defined our custom BERT and RoBERTa models introduced in the previous section for the data modeling part. We first loaded the pre-train model, then added an LSTM layer and a final linear layer under it. The input of the LSTM layer is the sequence output of the pre-trained model, extracting the first token's embeddings. The sequence output has the shape of (batch size, sequence length, hidden size), and the number of final linear layer output features depends on the number of labels of our dataset. Second, we defined AdamW as an optimizer, the Cross-Entropy Loss function as the criterion for the binary and multiclass classification tasks, and the Mean Squared Error Loss function as the criterion for the regression task (STS-B). We also defined a learning rate scheduler, and the number of training steps is the number of epochs we defined multiply the length of the training data loader. Last, we moved our model to Cuda so that the computation could get the GPU acceleration.

Third, we used a progress bar to show the model training and evaluation process. First, the training loader loads each batch of data to the model in every epoch. Then the model computes the loss and optimizes it. In addition, for the STS-B task, we need to unsqueeze the labels to change the tensor size from [batch size] to [batch size,1], which can have the same tensor size with the model outputs. Finally, the model predicts the validation dataset to evaluate our model depending on different tasks. Here is the evaluation method table.

| Task | Metric |
|---|---|
| CoLA | Matthew's Corr |
| SST-2 | Accuracy |
| MRPC | F1 & Accuracy |
| STS-B | Pearson & Spearman Corr |
| QQP | F1 & Accuracy |
| MNLI | Accuracy |
| QNLI | Accuracy |
| RTE | Accuracy |
| WNLI | Accuracy |

Where $MCC = \frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$, $ACC = \frac{TP+TN}{Total}$, $F1 = \frac{2TP}{2TP+FP+FN}$;
And Pearson correlation coefficient: $\rho X, Y = \frac{cov(X,Y)}{\sigma X \sigma Y}$,
Spearman's rank correlation coefficient: $r_s = \rho R(X), \rho R(Y) = \frac{cov(R(X),R(Y))}{\sigma R(X) \sigma R(Y)}$.

## 5. Hyper-parameters.

**Batch-size**:
the number of samples that will be propagated through the network.

| Task | Batch size |
|---|---|
| CoLA | 64 |
| SST-2 | 64 |
| MRPC | 16 |
| STS-B | 8 |
| QQP | 32 |

| | |
|---|---|
| MNLI | 32 |
| QNLI | 32 |
| RTE | 16 |
| WNLI | 4 |

**Epoch**:
indicates the number of passes of the entire training dataset the machine learning algorithm has completed
Epoch = 5 for all tasks except Cola and Wnli whose epoch = 30, since these two tasks have low accuracy. We increase the epoch to increase the times that we change the weights of the network. Then, the same number of times weights are changed in the neural network and the boundary goes from underfitting to optimal to overfitting.
**Learning rate**:
a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function
optimizer = AdamW(model.parameters(), lr=5e-5)
The learning rate that we set is low. Although our training will progress slowly, loss function can move toward to be optimal

## 6. Results.

| Task | Batch size | Epoch | Learning rate | BERT | RoBERTa |
|---|---|---|---|---|---|
| CoLA | 64 | 30 | 5e^5 | Matthew's Corr: 0.5931 | Matthew's Corr: 0.6131 |
| SST-2 | 64 | 5 | | Accuracy: 0.9243 | Accuracy: 0.938 |
| MRPC | 16 | 5 | | F1: 0.9091<br>Accuracy: 0.8701 | F1: 0.927<br>Accuracy: 0.8995 |
| STS-B | 8 | 5 | | Pearson: 0.8780<br>Spearman: 0.8757 | Pearson: 0.90488<br>Spearman Corr: 0.902 |
| QQP | 32 | 5 | | F1: 0.8789<br>Accuracy: 0.9095 | F1: 0.885<br>Accuracy: 0.9144 |
| MNLI | 32 | 5 | | Accuracy: 0.8338 | Accuracy: 0.865 |
| QNLI | 32 | 5 | | Accuracy: 0.9028 | Accuracy: 0.9218 |

| | | | | | |
|---|---|---|---|---|---|
| RTE | 16 | 5 | | Accuracy: 0.6245 | Accuracy: 0.711 |
| WNLI | 4 | 30 | | Accuracy: 0.5493 | Accuracy: 0.577 |

From the result, we can find that the BERT algorithm and RoBERTa algorithm are good for SST-2 task, MPRC task, QQP task and QNLI task especially. For these tasks, we all get high scores from our models. Comparing the results we get from the BERT model and RoBERTa mode, we can find that for each task, RoBERTa mode always has a better result, although the difference is not much. As mentioned, to improve the training procedure, RoBERTa removes the Next Sentence Prediction (NSP) task from BERT's pre-training and introduces dynamic masking so that the masked token changes during the training epochs. Moreover, RoBERTa uses larger text for pre-training. Therefore, we can conclude that RoBERTa model outperforms the BERTA model.

## 7. Summary and conclusions.

In the project, we learn how to build the architecture for Bert and RoBerta and analyze natural language through nine tasks. For each task, we use different metrics to evaluate their performance. When wIn the project, we learn how to build the architecture for Bert and RoBerta and analyze natural language through nine tasks. For each task, we use different metrics to evaluate their performance. When we run the code, we also met some problems due to large size of the task, so we change hyperparameters and run in GPU. From metrics, we conclude that Bert and Roberta both work well for nine tasks and Roberta works better. There are still many transformers which have similar functions. It is also interesting for us to explore in the future.e run the code, we also met some problems due to large size of the task, so we change hyperparameters and run in GPU. From metrics, we conclude that Bert and Roberta both work well for nine tasks and Roberta works better. There are still many transformers which have similar functions. It is also interesting for us to explore in the future.

## 8. References.

1. Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2019, February 22). *Glue: A multi-task benchmark and analysis platform for natural language understanding*. arXiv:1804.07461. Retrieved December 7, 2021, from https://arxiv.org/abs/1804.07461.

2. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz M., Davison J., Shleifer, S., Platen, P.V., Ma, C., Jernite Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M. (2020, October). Transformers: State-of-the-Art Natural Language Processing, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38-45). Association for Computational Linguistics. Retrieved December 7, 2021, from https://www.aclweb.org/anthology/2020.emnlp-demos.6.

3. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova (2019, May 24). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Retrieved December 7, 2021, from https://arxiv.org/pdf/1810.04805.pdf

4. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov (2019, July 26). RoBERTa: A Robustly Optimized BERT Pretraining Approach. Retrieved December 7, 2021, from https://arxiv.org/pdf/1907.11692.pdf

We borrowed and adjusted the codes from:

1. NLP lecture codes, https://github.com/amir-jafari/NLP/blob/master/Lecture_09/Lecture%20Code/12-training.py

2. Hugging Face transformers notebooks, https://colab.research.google.com/github/huggingface/notebooks/blob/master/examples/text_classification.ipynb#scrollTo=7k8ge1L1IrJk

3. Ashwin Geet D'Sa's answer on Stackoverflow, https://stackoverflow.com/questions/65205582/how-can-i-add-a-bi-lstm-layer-on-top-of-bert-model?rq=1