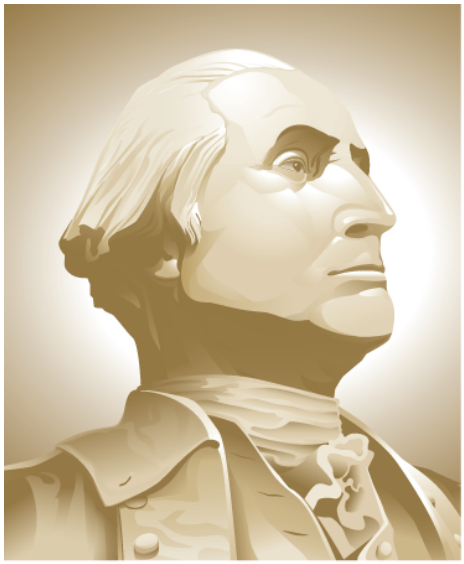


THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Final Project Report: GLUE

Course: Natural Language Processing

Professor: Dr. Amir Jafari

Authors: Tristin Johnson, Robert Hilly, Divya Parmar

December 9th, 2021

Introduction

For our final project, we decided to work on the General Language Understanding Evaluation (GLUE) benchmark, a collection of resources for training, evaluating, and analyzing natural language understanding systems. The GLUE benchmark includes 11 different datasets, consisting of sentence or sentence-pair language understanding tasks. These tasks are built on established, existing datasets that cover a wide range of typical natural language processing tasks, such as sentiment analysis, textual similarity, question pairs, and textual entailment.

We decided to work on the GLUE benchmark as the goal of this benchmark is consistent with the learning objectives of natural language processing: “The ultimate goal of GLUE is to drive research in the development of general and robust natural language understanding systems.”

Throughout the report, we provide an overview of each of the datasets, the data preprocessing steps performed on each dataset, the models used for each dataset, and each model’s performance on each dataset.

GLUE Tasks

The first task of the GLUE benchmark is the **Corpus of Linguistic Acceptability (CoLA)**. This dataset of 10,000 examples consists of English acceptability judgments drawn from books and journal articles on linguistic theory. Linguistic theory aims to explain the nature of human language in terms of basic underlying principles. In CoLA, the goal is to train a model that correctly classifies whether a given sentence is grammatically correct or not.
The structure of the dataset is quite simple as it contains a sentence and a label indicating whether or not the sentence is grammatically correct.

The next task is called the **Stanford Sentiment Treebank (SST)**. This dataset contains over 68,000 sentences from multiple movie reviews and human annotations of each review's sentiment. Therefore, the goal with this task is to train a model that correctly classifies the sentiment of a given movie review as either positive or negative. Sentiment analysis is a popular form of analysis, especially when it comes to companies who sell a product and want to know the feedback from the customer. Moreover, sentiment analysis scales well when we need to classify the sentiment of large amounts of textual data - a task that would prove time consuming if done manually by humans. The structure of SST is also fairly simple, as it contains a movie review and an associated label, indicating whether the movie review is positive or negative.

Another task in the GLUE benchmark is **Quora Question Pairs (QQP)**. The dataset consists of over 100,000 pairs of questions asked on popular Q&A website, Quora. Each question-pair is labeled by humans, indicating whether the question-pair has the same meaning or not. Since people tend to ask similarly worded questions, it can increase the time it takes for people to search for an answer to their question if they have to sift through multiple versions of the same question. Therefore, training a model that is capable of identifying whether pairs of questions have the same meaning or not has utility in terms of searching for answers.

The **Semantic Textual Similarity Benchmark (STS-B)** is a selection of datasets containing text data from image captions, news headlines, and user forums. Similar to QQP, the data is structured with a pair of sentences and a label indicating the similarity of the sentence-pair on the interval [1, 5]. However, instead of classifying whether a sentence pair is {similar, not similar}, we are quantifying the similarity between

The **STS Benchmark** is similar to QQP in that the data we are given are two sentences and a label showing how similar each sentence is on an interval [1, 5]. However, instead of classifying whether a pair of questions has the same meaning or not, we are interested in quantifying the similarity between a sentence-pair.

Next up is the **Microsoft Research Paraphrase Corpus (MRPC)**, which contains 5800 pairs of sentences that were scraped from news sources and then manually labeled by humans to determine whether each pair of sentences has paraphrase/semantic similarity. Here, our goal is to build a classifier that can distinguish whether each sentence-pair shares the same semantic content or not.

The next dataset is called **Recognizing Textual Entailment (RTE)**. This dataset is a combination of 4 other RTE datasets and contains over 5,500 examples. The goal of textual entailment recognition is to train a model that takes two text fragments and sees whether the meaning of one text fragment can be entailed/inferred from the other text fragment. The structure of RTE is slightly more complex compared to the previous datasets in that we have two columns of text fragments and another column indicating whether the text fragments have textual entailment.

Another task within the GLUE benchmark originated from the **Winograd Schema Challenge**, which is a reading comprehension task in which a system must read a sentence with a pronoun and select the referent of that pronoun from a list of choices.

However, the GLUE benchmark modifies the goal of this challenge. Instead of creating a model that reads a sentence with a pronoun and chooses a referent of that pronoun from a set of choices, the goal is to create a model that predicts whether a given sentence with a substituted pronoun can be inferred from the original sentence.

Because of this change, the task is now called **Winograd Natural Language Interference (WNLI)**. The dataset contains over 800 examples, in the form of two sentences and a label. Here, each example is evaluated separately so that there is no systematic correspondence between the model's score.

Next, we have **Multi-Genre Natural Language Interface Matched (MNLI-m)**. This dataset contains ~400,000 train observations in the base MNLI and ~9000 validation examples in MNLI-m. Each observation contains a premise ("Your gift is appreciated by each and every

student who will benefit from your generosity.") and a hypothesis ("Hundreds of students will benefit from your generosity."). Here, we're interested in predicting if the hypothesis logically flows from the premise. Each pair is either classified as entailment (it does flow), neutral, or contradiction (premise shows the opposite of hypothesis).

As a continuation to the prior task, we have **MNLI Mismatched (MNLI-mm)**. This task is the exact same as the one prior, except it uses mismatched samples to test on. The model is trained on the same dataset as the prior task, so no new training is needed here. We will use ~9000 new validation samples for this task.

Next we have **Diagnostics (AX)**. This task also contains sentence pairs in which we must check for entailment. Therefore, the model trained in MNLI-m can be applied to a new validation set of ~1,100 records. For this task, Matthew's Correlation is applied instead of accuracy.

Finally, we have **Question Natural Language Interface (QNLI)**. This dataset is derived from the Stanford Question Answering Dataset (SQuAD). This task contains a question and a sentence taken from a context paragraph, with the intent to see if the sentence answers the question. We will use ~100,000 train pairs and ~5,000 validation pairs.

Data Preprocessing

To complete each of the GLUE tasks, we used a combination of pre-trained Transformer models from HuggingFace and PyTorch. When applying the models to each dataset, we made some changes regarding the data preprocessing in order to correctly fit each model and standardize the input into a neural network. Moreover, we fit an LSTM model to SST and CoLA to compare it to the performance of the pre-trained Transformer models.

Before applying the LSTM to SST and CoLA, we first removed special characters, whitespace, digits, and stopwords from each dataset. After cleaning each dataset, we created a

one-hot encoded dictionary from the text in each dataset. One-hot encoding allows us to convert text to an array of numbers, while also keeping the integrity of each sentence's information. This process was applied to both the training and testing sets, in order to maintain the integrity of the results of the model.

Finally, we padded each sentence so that each input given to the LSTM is the same length. Now that the data is standardized and encoded, we can create a PyTorch Data Loader for both the training and testing sets.

The preprocessing steps are slightly different when we apply Transformer models. First, we pull in the pre-trained model's tokenizer and apply it to the given dataset in GLUE. This results in each token being assigned an ID, which we can feed to our models. However, this process does not account for the length of each input. For example, the SST task has over 67,000 examples of reviews, and not all of them are the same length. Therefore, we can add attention masks to the array. An attention mask is a binary tensor indicating the position of the padded indices so that the model does not attend to them, such as a 1 indicating that a value should be attended to while a 0 indicates a padded value. Adding attention masks and padding each sequence results in each input being the same length, while still holding the same information in order to predict the label.

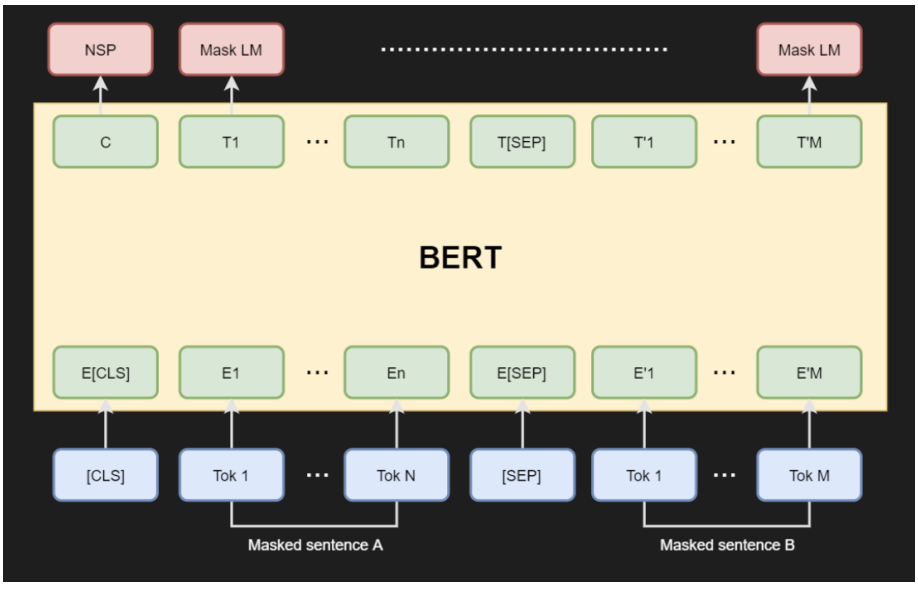
If you look at our code, you'll notice that how this preprocessing is done is different depending on whether we used HuggingFace or PyTorch. For the former, we can perform preprocessing on the HuggingFace Dataset and feed it to our pre-trained model. For the latter, we need to convert our inputs to a Tensor Dataset and use a DataLoader to iterate over the training and testing sets.

While we did experiment with rolling up our own models, using pre-trained models has several advantages. For example, we save time, money, and computational resources as we don't have to train our models from scratch. Moreover, we can use the knowledge stored in pre-trained language models (BERT, ALBERTA, XLNet, etc) and apply them to the tasks we are

interested in, whether that be classifying duplicate question-pairs or classifying the sentiment of movie reviews.

Models

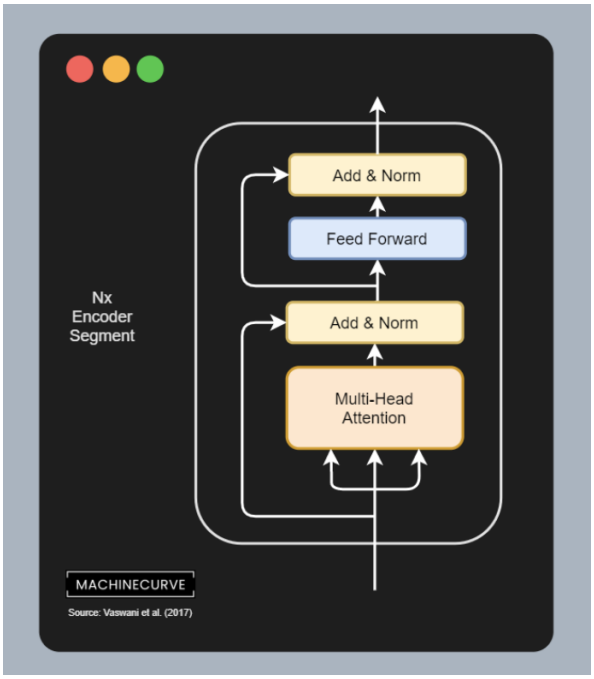
Given the diversity of tasks that comprise GLUE, we fit different models to each of the tasks. For SST, CoLA, WNLI, and RTE, we used the transformer models ALBERT and ELECTRA. ALBERT is a lite version of BERT, as it has less trainable parameters than its larger counterpart. Indeed, BERT is a very large transformer, with 110 million trainable parameters in its original form and 340 million trainable parameters in its large form. Therefore, training ALBERT results in less memory consumption and an increase in training speed. To better understand the changes that ALBERT makes, let's look at the BERT Architecture:



BERT takes two sets of tokens as inputs and tokenizes them. Next, a classification (CLS) token is added, which contains sentence-level information on the text. The tokens are then fed into BERT, where word embedding occurs.

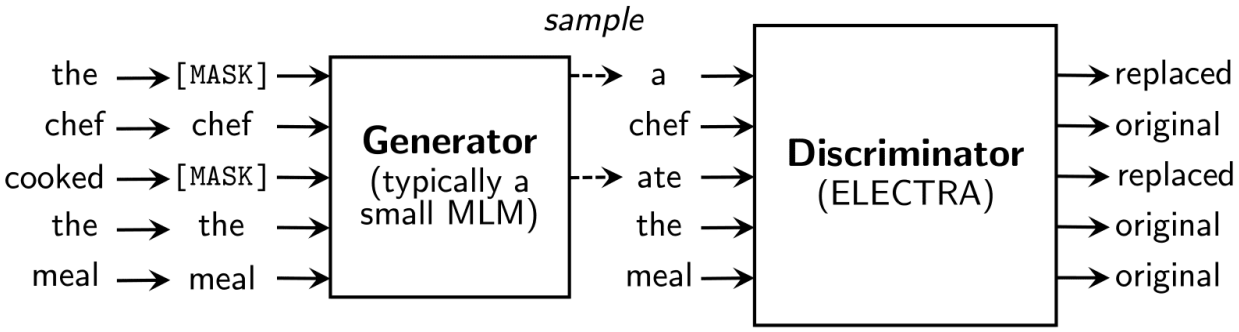
BERT utilizes two language tasks - a Masked Language Model for predicting output tokens, and a Next Sentence Prediction Model to learn sentence-level information for the next

input. Now that we have an understanding of the architecture of BERT, we can look at the architecture of ALBERT:



Here, the architecture of ALBERT is just the encoder segment from BERT with two changes. The first change is that the embeddings are *factorized*, meaning that the parameters of the embedding and the hidden state size are decomposed into two smaller matrices. The second change is that ALBERT applies cross-layer parameter sharing, meaning that the parameters of the “Multi-Head Attention” encoded segment and the Feedforward encoded segment are shared. These changes make ALBERT more efficient than BERT.

The next model that was tested on four of the tasks is ELECTRA. The ELECTRA model is inspired by Generative Adversarial Networks (GANs), where training involves two Transformer models - a generator and a discriminator (NOTE: the generator and the discriminator share the same input word embeddings:



In the diagram above, the generator replaces tokens and is then trained as a Masked Language Model (MLM). Here, the generator replaces “the” with “a” and “cooked” with “ate”. The discriminator then tries to identify which tokens were replaced by the generator in the sequence. After pre-training, the generator is dropped from the model and the discriminator is fine-tuned on downstream tasks.

We also looked at the DistilBERT architecture, which is a smaller version of BERT. In "DistilBERT, a distilled version of BERT: smaller, faster, cheaper, and lighter", Sanh et. al from HuggingFace show that, "[...] it is possible to reduce the size of a BERT model by 40%, while retaining 97% if its language understanding capabilities and being 60% faster" (Sanh et. al 2020). Therefore, with DistilBERT, we’re able to efficiently train a language model using a single GPU.

The DistilBERT architecture is quite similar to the BERT architecture. However, in DistilBERT, "The token-type embeddings and pooler are removed while the number of layers is reduced by a factor of 2" (Sanh et. al 2020). We applied DistilBERT to QQP, MRPC, and SST-B.

The next model that was tested on SST and CoLA was a customized LSTM model. An LSTM is a type of Recurrent Neural Network (RNN), but performs a lot better than traditional RNNs in terms of remembering past information. This is because LSTMs utilize a hold over in memory, by memorizing certain patterns. This allows LSTMs to outperform general RNNs in natural language understanding.

LSTMs can have multiple hidden layers. As information passes through each layer, the relevant information is kept and all the irrelevant information gets discarded Below, is the LSTM architecture we used:

```
SentimentAnalysisLSTM(
  (embedding): Embedding(1501, 64)
  (lstm1): LSTM(64, 256, num_layers=2, batch_first=True)
  (lstm2): LSTM(64, 256, num_layers=2, batch_first=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (linear): Linear(in_features=256, out_features=1, bias=True)
  (act): Sigmoid()
)
```

The first thing that is defined in the architecture is the embedding layer. The embedding layer represents words in a dense vector representation, where the position of a word within the vector space is based on the words that surround the word when it is used.

Along with the embedding layer, the vocabulary size from our corpus is provided. Next, is two LSTM layers. These both consist of the embedding dimension, which is 64, along with the number of neurons, which is 256. Then we added a dropout of 0.3 to help reduce overfitting, a linear layer to flatten the output, and sigmoid as the activation function.

Experimental Setup

As mentioned earlier, all of the tasks were trained and tested using PyTorch as the framework. PyTorch's framework provides lots of flexibility when it comes to training and testing a model, which is why we used this framework. Furthermore, we will be using PyTorch's Data Set and Data Loader to convert all of the encoded datasets from Numpy arrays to Tensors. When it comes to the transformers, we used the 'transformers' package from Hugging Face to load each of the models tokenizer along with the models architecture.

When judging the performance of each model, each GLUE task has a different metric that is used to look at the performance. For CoLA and Diagnostics Main, Matthew's Correlation

coefficient is the metric. For MRPC and QQP, both an F1-Score and accuracy is used as the metric. For STSB, Pearson-Spearman Correlation Coefficient is used as the metric. And lastly, for SST, MNLI-Matched, MNLI-Mismatched, QNLI, RTE, and WNLI, accuracy is the metric used to judge model performance.

With regards to model hyperparameters, each dataset's parameters varied depending on the model used and the dataset itself. Applying a custom transformer model using ALBERT and ELECTRA on SST, RTE, WNLI, and CoLA, the batch size was 32, the maximum length of encoded tensors was 512, the learning rate was 0.001 with 5 epochs each. The optimizer was AdamW, the learning rate scheduler was 'ReduceLROnPlateau', and the loss function was CrossEntropyLoss. For the custom LSTM that was applied to CoLA and SST, the model hyperparameters included a batch size of 32, a learning rate of 0.001, a maximum length of 128 encoded tensors, 1 output dimension, an embedding dimension of 64, a hidden dimension of 256, and 20 epochs. Furthermore, the optimizer for the LSTM was Adam, the learning rate scheduler was 'ReduceLROnPlateau', and the loss function was Binary Cross Entropy loss.

All of these model parameters were adjusted and updated throughout training a model on each dataset, however, all of the above parameters were the ones that worked best and were most consistent with results in terms of the models performance. On the other hand, thus far we know what each dataset consists of, the preprocessing within each task, an explanation of each model that we used, and the hyperparameters used for each task. Knowing all of this, we can now train and test each model within each task, and take a look at the results of each GLUE task.

Performance Judging

When it comes to calculating the accuracy of our models on the GLUE tasks, each dataset has a different metric used to judge the performance of each model. There are four

different metrics, including accuracy, F1-Score, Pearson's Coefficient, Spearman's Coefficient, and Matthew's Correlation Coefficient.

Accuracy is defined as the percentage of observations a given classifier correctly classifies. For QQP, the objective is to create a model that is able to distinguish between whether a given pair of questions have the same meaning or not. Say we have 10 observations, where the first 5 have the same meaning and the other 5 have a different meaning. Our model classifies 4 of the first five observations as having the same meaning and one of them as not having the same meaning. For the second half, it classifies 4 of the observations as not having the same meaning and 1 of them as having the same meaning. Given this, our model correctly classifies 8/10 of the observations, resulting in an accuracy of 80%.

The **F1-score** is the average between a classifier's precision and recall. One area this metric is useful in is for evaluating classifiers that are trained on imbalanced datasets. For example, if our target feature for QQP is 90% duplicate questions and 10% non-duplicate questions, we could simply create a classifier that classifies all observations as duplicate questions and obtain 90% accuracy. However, such a model would have a low F1-score since it doesn't attempt to discern between duplicate and non-duplicate questions.

Pearson's correlation measures the linear association between two variables, X and Y, and is bounded on the interval [-1, 1]. Here, -1 indicates perfect anti-correlation, while 1 indicates perfect correlation. For our purposes, we want the correlation between our predictions and the target values to be close to 1.

Spearman's correlation is slightly different from Pearson's correlation in that instead of measuring the linear association between two variables, it measures the monotonicity between two variables and is also bounded on the interval [-1, 1]. Similar to Pearson's we want this quantity to be close to 1.

Matthew's Correlation Coefficient is a more reliable statistical rate which produces a high score only if the prediction obtained good results in all of the four confusion matrix

categories (true positives, false negatives, true negatives, and false positives), proportionally to both the size of positive elements and the size of negative elements in the dataset. This metric was used in two of the tasks, in which the data in these tasks are imbalanced and is typical of a model to make guesses on predictions that just so happen to be correct.

Results

Below, are our results for each task:

	DistilBERT
Accuracy (Test)	85.5%
F1-Score (Test)	0.90

SST-B:

	DistilBERT
Pearson (Test)	87.3%
Spearman (Test)	86.9%

QQP:

	DistilBERT
Accuracy (Test)	88.2%
F1-Score (Test)	0.843

Regarding the CoLA dataset, the testing score was based on Matthew's Correlation Coefficient (MCC). After applying this dataset to ELECTRA, ALBERT, custom ELECTRA, custom ALBERT, and LSTM, the results can be seen below:

CoLA	ELECTRA	ALBERT	Custom ELECTRA	Custom ALBERT	LSTM
MCC (Test)	0.5508	0.4187	0.212	0.1981	0.11951

The results show that ELECTRA was by far the best model in terms of performance, with ALBERT coming in second. Furthermore, the training time for ELECTRA was only 3 minutes and 52 seconds per epoch, compared to ALBERT's 11 minutes and 17 seconds. Next, we can take a look at the results from SST:

SST	ELECTRA	ALBERT	Custom ELECTRA	Custom ALBERT	LSTM
Accuracy (Test)	90.137%	86.811%	51.927%	50.817%	77.315%

Once again, the results show that ELECTRA was the most accurate for sentiment analysis, with ALBERT coming second. Furthermore, the LSTM model was able to achieve just over 77%, and only taking 52 seconds per epoch when training, which is quite impressive. Next, we can look at the results from WNLI:

WNLI	ELECTRA	ALBERT	Custom ELECTRA	Custom ALBERT
Accuracy (Test)	56.828%	57.38%	56.338%	56.338%

Looking at these results, ALBERT was able to achieve the highest accuracy, with all the other models just behind. What's interesting here is that the custom ELECTRA and ALBERT models got the exact same accuracy score, with ELECTRA only half a percent ahead. Next, we can look at the results from RTE:

RTE	ELECTRA	ALBERT	Custom ELECTRA	Custom ALBERT
Accuracy (Test)	63.176%	54.151%	52.708%	51.818%

Once again, the ELECTRA model outperformed all other models by almost 10%, and only took around 58 seconds per epoch compared to ALBERT's 2 minutes and 40 seconds at only 54%. Looking at all the results from ELECTRA and ALBERT, it is quite clear that these pretrained models perform best when they are not configured, and use all of their predetermined parameters. On the other hand, the LSTM model was able to hold on it's own and output some impressive results, considering how simple the network is.

MNLI and Diagnostics

Task	**BERT**
MNLI Matched	35.45% (Accuracy)
MNLI Mismatched	35.22% (Accuracy)
Diagnostics	N/A

All three of the tasks above were trained in the same script, and thus share the same training model and transformer (BERT). The training was done in 2 epochs, with a learning rate of 0.0005 and a batch size of 8. The training set had to be reduced in size to 8,000, as the original size of nearly 400,000 records was causing memory errors on our GPU.

This model did not perform well, finishing below just guessing the most common class (which would obtain 36.5% on Matched and 35.6% on Mismatched). However, this was due to the very small number of samples taken and epochs run due to computational limitations. With more

time, computational power, and transformer testing, there is potential to build a higher performing model.

Diagnostics came with only -1 for the labels, which is not correct. Therefore, we chose to ignore it for our metrics.

QNLI

	BERT
Accuracy (Validation)	79.736%

Initially, training this model was leading to CUDA out-of-memory errors. However, by reducing the train sample size from ~100,000 to 18,000 and using a small batch size of 4, we were able to get the training and validation to run through.

Using a BERT transformer and a starting learning rate of 4.7e-05, the model performed well. The TrainingArguments function modified the learning rate during partial epochs (reducing it slightly every 0.2 epoch). The accuracy increased from 78.784% on the first epoch to 79.736% on the second epoch. With more time and processing power, there is potential for further improvement.

Conclusion

Overall, we were content with the results we received while completing each of the 11 GLUE tasks. From applying the BERT, DISTILBERT, ALBERT, ELECTRA and LSTM models to the datasets, we learned a lot about each of the models along with how each model performs against each other given the same task. Furthermore, it is safe to say that DistilBERT and

ELECTRA were our two most promising models as they reported the highest and most consistent levels of performance throughout the GLUE Benchmark. In the future, we would like to apply even more models to each of the tasks, and start to fine-tune the models that outperform others in specific natural language understanding problems, as this analysis would help with model decisions in the real world. All in all, we look forward to applying the knowledge we gained from this project to future datasets, and now have a better understanding of dealing with natural language processing tasks.

References

1. “Transformers.” *Hugging Face*. https://huggingface.co/docs/transformers/index

2. “Sentiment Analysis using LSTM – PyTorch”. *Kaggle,* 8 June 2021, https://www.kaggle.com/arunmohan003/sentiment-analysis-using-lstm-pytorch

3. “Text Classification on GLUE”. *Google Colab*. https://colab.research.google.com/github/huggingface/notebooks/blob/master/examples/text_classification.ipynb#scrollTo=HFASsisvIrIb

4. Verma, Dhruv. “Fine-tuning Pre-Trained Transformer Models for Sentence Entailment.” *Towards Data Science,* 14 Jan. 2021. https://towardsdatascience.com/fine-tuning-pre-trained-transformer-models-for-sentence-entailment-d87caf9ec9db

5. Shekhar, Shraddha. “LSTM for Text Classification in Python.” *Analytics Vidhya,* 14 June 2021. https://www.analyticsvidhya.com/blog/2021/06/lstm-for-text-classification/

6. Clark, Kevin. “ELECTRA.” *GitHub*. https://github.com/google-research/electra

7. Versloot, Christian. “ALBERT explained: A Lite Bert.” *Machine Curve,* 6 Jan. 2021. https://www.machinecurve.com/index.php/2021/01/06/albert-explained-a-lite-bert/

8. “Sanh et. al” "DistilBERT, a distilled version of BERT: smaller, faster, cheaper, and lighter." Mar. 2020. https://arxiv.org/pdf/1910.01108.pdf

9. “HuggingFace NLP Course” https://huggingface.co/course/chapter1/1?fw=pt

10. Bowman, Sam. “MultiNLI.” *Multinli*, https://cims.nyu.edu/~sbowman/multinli/.

11. Horev, Rani. “Bert Explained: State of the Art Language Model for NLP.” *Medium*, Towards Data Science, 17 Nov. 2018, https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

12. Lutkevich, Ben. “What Is Bert (Language Model) and How Does It Work?” *SearchEnterpriseAI*, TechTarget, 27 Jan. 2020, https://searchenterpriseai.techtarget.com/definition/BERT-language-model.