Natural Language Processing - Final Project Report

Harish Ram, Zhuohan Yu, Sisi Zhang

Introduction

In order to properly build a model to accurately perform multiple tasks, we used the GLUE (General Language Understanding Evaluation) benchmark to test the model performance. GLUE consists of several tasks including single sentence, similarity and paraphrase and inference tasks with a unique dataset for each task. Our group decided to test using LSTM and several transformers on the GLUE benchmark to determine what we should focus on throughout this project.

Using the transformers repository hosted on the huggingface Github, our group analyzed the metrics of using BERT, XLNet and ELECTRA transformers on every GLUE task. After fine-tuning each model and demonstrating the effectiveness of its architecture on each task, ELECTRA performed the best individually on most of the tasks and had the highest average benchmark among all of the transformers we used. Thus, looking into the framework and layers of the ELECTRA model, it was important to understand that the pre-training method was distinctly different from BERT and XLNet (which is based on the BERT architecture). As opposed to Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) which is the basis of the BERT framework, ELECTRA uses Replaced Token Detection which is computationally more efficient without sacrificing performance.

Once we decided to focus on ELECTRA, we began fine tuning parameters such as sequence length, batch size, learning rate, position embedding type, hidden size and number of attention heads to achieve higher metrics on the GLUE benchmark. Along with hyper parameter tuning, we also sought to test out multiple ELECTRA models such as electra-discriminator and electra-generator as well as sizes - small, base and large to increase our performance. Our goal in this process was to establish whether or not certain model types within ELECTRA worked better for different tasks.

Once we had the highest benchmark for each task, our next objective was to ensemble the ELECTRA model with other transformer models to increase our score.

Description of the Data

The GLUE dataset comes from multiple sources and concerns varying topics all involving natural language processing. The overall tasks that GLUE contains can be either single sentence tasks, similarity and paraphrase tasks and inference tasks. There are two datasets that both concern single sentence tasks - the first being: The Corpus of Linguistic Acceptability (CoLA) which involves deciphering whether a sentence is grammatically acceptable. The second task is the Stanford Sentiment Treebank (SST-2) which requires using sentiment

analysis on a set of movie reviews to predict whether the review was generally positive or negative.

The similarity and paraphrase tasks are done on several datasets. The first task in this section uses the Microsoft Research Paraphrase Corpus (MRPC) which contains sentence pairs from online news sources in which the task is to determine semantic equivalence. Next, the Quora Question Pairs (QQP) dataset is a question-answer pair corpus coming from the online forum website, Quora to check if two questions are similar. The final dataset in this category comes from the Semantic Textual Similarity Benchmark (STS-B) which is a collection of sentence pairs pulled from news sources and media captions with a corresponding similarity score from 1-5.

The first dataset used for the inference task is in the form of sentence pairs (the first sentence is a premise and the second is a hypothesis) sourced from Multi-Genre Natural Inference Corpus MNLI) which contains speech, fiction and government reports. The goal is to see whether the premise entails, does not entail or has no relation to the hypothesis (neutral). The next dataset, coming from The Stanford Question Answering dataset (QNLI) is a question-answer pair in which the answer sentence must be extracted from a corresponding paragraph. The answer may or may not be an accurate answer, so the task is to evaluate its accuracy. The Recognizing Textual Entailment (RTE) dataset comes from Wikipedia and news text with the task being to determine whether the meaning or context of a sentence fragment can be inferred from the other. Finally, the last task comes from the Winograd Schema Challenge (WNLI). This GLUE task uses sentence-word pair dataset in which the word is a pronoun that refers to a word in the sentence. In order to complete this task, we must first understand the context of the sentence and accurately determine where the pronoun should be placed for the context of the sentence to still make sense.

Model Description

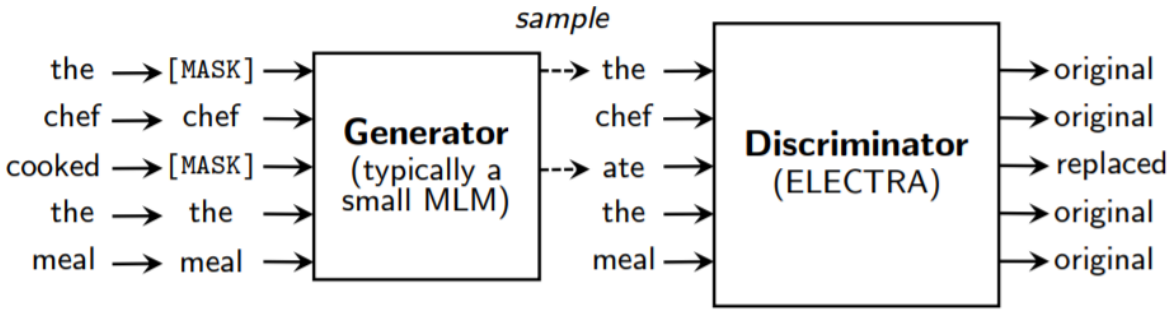
ELECTRA

The model we are using for this project will be ELECTRA. Using the BERT model framework as a comparison, ELECTRA works in a computationally more efficient way. As opposed to doing Masked Language Modeling (MLM) wherein 15% of the tokens in the data are masked and learned, ELECTRA uses a technique called Replaced Token Detection which builds on the generator (or any type of model that produces an output distribution over tokens) and then trains a discriminator to predict whether the token was replaced by the generator or not.

The reason that we are using ELECTRA is for its different architecture and function over BERT-based transformers as well as its highly computationally-efficient nature. The primary difference is that ELECTRA uses Replaced Token Detection as well as a generator model during pre-training.

Replaced Token Detection

In the figure below, the generator and discriminator in ELECTRA are trained jointly. The generator is trained to replace tokens in the sequence typically using an MLM while the discriminator determines whether every single token in the training corpus is either from the original corpus or replaced by the generator. By using both a generator and a discriminator, the model can learn from all the tokens during pre-training as opposed to only 15% like in a BERT model.



The following equation is how the discriminator determines whether the token is from the original dataset or if it was replaced by the generator. The discriminator assigns a value to each token depending on whether it believes the token is “corrupted” or replaced by the generator and passes these values through a sigmoid output layer.

$$D(\boldsymbol{x}, t) = \text{sigmoid}(w^T h_D(\boldsymbol{x})_t)$$

The core functionality of the ELECTRA model is defined by the four equations below. The generator first selects a random number of positions throughout the training corpus to mask and subsequently replaced by a [MASK] token. The generator then learns from the masked tokens, but before the discriminator begins learning, the generator replaces those masked tokens with possible alternatives. After which, the discriminator learns to decipher the corrupted tokens from the original tokens.

$$m_i \sim \text{unif}\{1, n\} \text{ for } i = 1 \text{ to } k \qquad \boldsymbol{x}^{\text{masked}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \texttt{[MASK]})$$

$$\hat{x}_i \sim p_G(x_i | \boldsymbol{x}^{\text{masked}}) \text{ for } i \in \boldsymbol{m} \qquad \boldsymbol{x}^{\text{corrupt}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \hat{\boldsymbol{x}})$$

GAN vs. Maximum Likelihood

A Generative Adversarial Network or GAN is a type of implicit generative model that uses unsupervised learning to pit two networks against each other to generate synthetic results. The GAN network is split into a generator and discriminator where both are multilayer perceptron networks and have different, but cooperative training procedures. The discriminator calculates

the probability the sample was calculated from the generator or from the actual corpus while the generator attempts to increase the probability the discriminator makes a mistake. This keeps working until the probability the discriminator calculates is equal to ½ for every single token. As a result, the generated tokens are very difficult to determine from the actual corpus. This system relies on backpropagation to reach the desired probability. In this case, backpropagation refers to iterating through the GAN and updating the probabilities of the discriminator based on the generator output to reach the desired probability as mentioned before.

However, there are some differences made to the Generative Adversarial Network used in ELECTRA. For example, in a GAN, when the generator replaces a token with the same word, the discriminator would consider that to be a “corrupt” or replaced token. However, in the ELECTRA framework, the discriminator would consider this a real token. This was implemented because the model would perform better on GLUE tasks.

Another difference is how GAN produces a low-entropy output distribution which means the diversity in the samples it produces is very low compared to Maximum LIkelihood which is able to produce more diverse samples by looking at the probability of the tokens in the sample space.

As a result of these differences, ELECTRA uses a manipulated version of GAN as a generator along with a discriminator while BERT purely uses MLM and NSP during pre-training.

Weight Sharing

Another aspect of ELECTRA that makes it unique compared to a BERT model, is its weight sharing in embedding weights. Embedding is a process of vectorizing a word or token. Embedded forms of a token take on a similar definition, contextual meaning and representation of the word. Thus, similar embedded words in the form of real-value vectors will be near one another in the same embedding space. In ELECTRA, the embedding weights of tokens are shared between the generator and discriminator during pre-training. This is generally because the size of the generator is typically ¼ or ½ the size of the discriminator; therefore, when the embedding weights are shared, all of the transformer weights can be tied as well. This is useful during training because the discriminator only updates the weights that are in the data or created by the generator, while the generator updates all tokens related to the vocabulary.

Comparison of BERT and ELECTRA

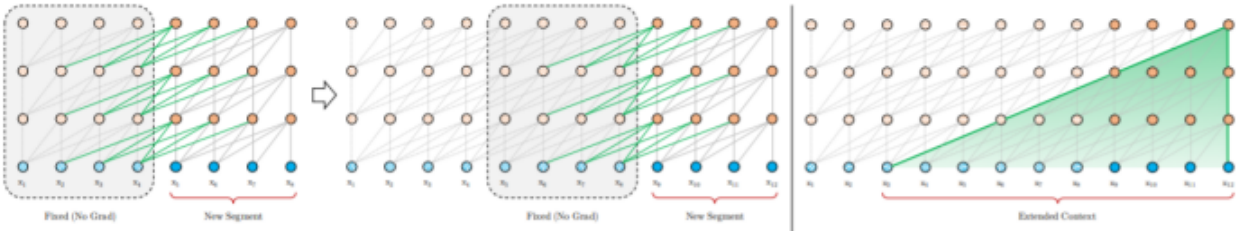
The primary difference between the BERT and ELECTRA model happens exclusively during pre-training. The ELECTRA model uses Replaced Token Detection, a Maximum-Likelihood enforced generator and weight sharing techniques during embedding during pre-training while the BERT model uses MLM and NSP. After pre-training is over, the ELECTRA and BERT model follow the same framework and architecture and perform the same training on inputs. The reason that ELECTRA is being used for this project is due to its higher benchmark values and faster computational speeds.

XLnet

XLnet is another model we used. Although its architecture is similar to that of Bert, there are many novel techniques it uses to improve on performance..

TransformerXL

TransformerXL is another model that was published before XLnet which solved the issue of limited fixed-length context in the setting of language modeling by using the Segment-Level Recurrence with State Reuse.



(a) Training phase. (b) Evaluation phase.

Instead of learning dependency on a fixed length, segment-level recurrence allows the model to fix the previous segment into cache and reuse it as an extended context in the new segment. And, the recurrent mechanism is also faster to evaluate, because the representations from the previous segments can be reused instead of being computed from scratch.This is the basic idea of XLnet.

Permutation Language Modeling

The aim of using Permutation Language Modeling is to combine the advantages of BERT and AR language modeling as well as avoid their weaknesses. Because the model will only encounter text sequences with the natural order during fine tuning, permutation modeling will keep the natural sequence order and rely on a proper attention mask in Transformers to achieve permutation of the factorization order. This kind of modeling will gather information from all positions on both sides and maximize the likelihood of sequence.

Two-Stream Self-Attention

XLnet used two hidden representations to solve contradictions caused by using target-aware representation in standard Transformer architecture. First is the content representation which encodes both the context and token, and serves a similar role to the standard hidden states in Transformer. The second is the query representation, which only has access to the contextual information and the position but not the token itself.

Experimental setup

Experimental Setup

The base code we are using to run the ELECTRA model on the GLUE tasks, called run_glue.py comes directly from the huggingface Github repository. The GLUE website provides the data already separated into a training and development set used during model training and a testing set used for model evaluation.

run_glue.py is a file that uses some fine-tuning libraries and allows us to use predefined code that preprocesses, formats, trains and evaluates any GLUE task we predefine. By doing this, we were able to specify the name of any transformer models we wanted to test. Furthermore, this allowed us to also tune several hyperparameters which we listed above in order to maximize our score, such as max_seq_length, batch size.

Also, we downloaded the config.json file from each pretrained models' website on hugging face. By specifying their local paths of config, we were able to tune more detailed parameters of each model in config.json, such as dropout rate. However, even though we changed several parameters, there are some others we cannot change. Since those models are pretrain models, they have to use the same weights, dict_states, and parameters as they were pretrained. Although this tuning didn't have a significant effect, it helped us better understand our models.

Besides, we also modified some pretrain models by ourselves to improve some tasks that have lower scores. For example, modifying electra_base_discriminator to predict CoLA. However, we didn't get a better result than the pretrained one. After that, we used logistic regression to ensemble models to have better performance and got our final results.

Performance Metrics

In order to judge the performance of the model that we have fine tuned, we will be using pre-specified metrics defined by the GLUE benchmark. The first task, CoLA involves Linguistic Acceptability, so the metric to evaluate performance is the Matthew's Correlation Coefficient (MCC) which is primarily used for binary and multiclass classification. This metric takes into account false positives and negatives as well as class imbalances. The value of MCC is -1 which indicates perfect prediction, 0 for an average random prediction and -1 for an inverse prediction. For a task involving determining whether a sentence is grammatical acceptability, the MCC metric allows for proper classification.

For the task involving Sentiment Similarity such STS-B in the GLUE benchmark, the Pearson-Spearman Coefficient is used because of its ability to look at the ranked values of each variable and evaluate relationships involving ordinal variables. Since the label column within this task involves a similarity value between 1-5, the Pearson-Spearman coefficient will be the most accurate performance metric.

The F1 and Accuracy score are both used for the MRPC and QQP task. MRPC involves determining semantic similarity between sentence pairs. Due to the fact that there is a class imbalance in the dataset in both of these tasks, using an F1 score along with an accuracy score is necessary. F1 scores are calculated by incorporating both precision and recall, so it takes into account the accurate predictions as well as other errors.

For the rest of the tasks on the GLUE benchmark, we will test performance of our model using a simple accuracy score. An accuracy score is used to determine whether the predictions made by the model are correct or incorrect. The tasks that use this will be QNLI, WNLI, SST, MNLI and RTE. These tasks involve predicting entailment or non-entailment and because the label classes are balanced, an F1 score is not required.

Hyper-Parameters

The ELECTRA models for numerous types of parameters that allow us to finetune and manipulate the intricacies of how the model handles the inputs. The parameters that we focused on first were maximum sequence length, batch size, learning rate and number of epochs.

The *batch size* of the model corresponds to the number of samples that the model should train before updating the weights and biases. As default, the model had a batch size of 32 which we tested out increasing and decreasing to evaluate the score. By increasing the batch size, there are fewer optimization steps, thus less training time. We found that decreasing the batch size by a factor of 2 increases the performance metric slightly in cases such as CoLA, SST-2, RTE and QNLI while also increasing the amount of time the model takes to run. However, since this was not the case for all tasks, we kept it at 32.

The *learning rate* is defined by the number of steps before the model weights change. In general, increasing the learning rate comes with the risk of overshooting meaning that the step sizes are too large in the direction of the minimized loss function and shoot past the minimum. On the other hand, the lower learning rate allows for smaller step sizes which causes the model to take a lot longer to reach minimized loss. In our case, we tested decreasing and increasing the learning rate by a factor of 10 for each task and the best learning rate we found was 2E-5.

The *maximum sequence length* was another parameter that we looked into that allows us to define the maximum sequence that is inputted into the training model at a time before the internal parameters are updated. According to the documentation of the ELECTRA model code, the default maximum sequence length is 128 while larger sequences will be truncated and smaller sequences will be padded. After tuning this parameter, we determined that a maximum sequence length that was doubled can increase the performance metric slightly. However, this is not consistent with all the tasks; as a result, we used a maximum sequence length of 128 for all tasks.

The *number of epochs* for our model is defined by how many iterations the learning algorithm goes through the training data. We defined our model to use 3 epochs for every single task. We

found that with fewer epochs, the accuracy score does not increase while an increased number of epochs did not help the score increase as well.

Results

Throughout this project, we focused on using various methods to output the highest scores in all the GLUE tasks. As a result, our project can be split up into 3 sections: getting the benchmark values on the transformers we focused on, outputting the predictions for several transformers to use for further analysis and finally using ensemble techniques to maximize the metrics on all the glue tasks.

Getting a Benchmark

The benchmark we chose was the score of the bert_base_cased model that we got by running the run_glue.py file. BERT is the default pretrained model run_glue.py used, and it is one of the most widely-used and earliest pre-trained transformer-based language models. Since it has pre-trained for representation learning, BERT has a better understanding of the meaning of language within context and is able to predict results for various types of natural language processing tasks due to its framework.

To briefly explain the architecture as reasoning behind using it as a benchmark as well as in our final ensemble model, BERT uses MLM and NSP during pre-training. Therefore, it is able to understand the context of a sentence bidirectionally by predicting tokens in the pre-training corpus as well as predicting whether the context of the next sentence makes sense in reference to the previous sentence.

The table below shows our findings for all the GLUE tasks:

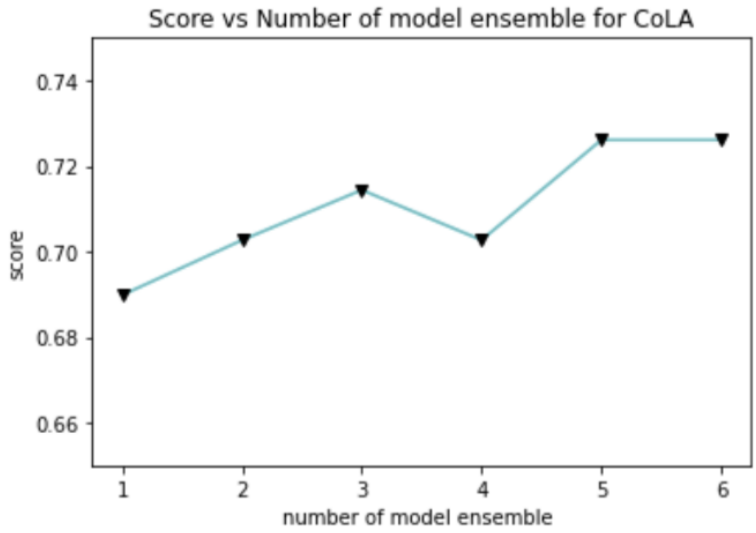
Task	Metric	Bert-base-cased	Xlnet-base-cased	Electra-base-discriminator
CoLA	Matthews corr	57.01	31.99	65.85
SST-2	Accuracy	92.43	94.15	95.41
MRPC	F1/Accuracy	89.46/84.8	89.76/85.29	90.88/86.76
STS-B	Pearson/Spearman corr.	88.82/88.5	86.73/86.56	90.31/90.35
QQP	Accuracy/F1	90.71/87.49	90.83/87.74	91.79/89.06
MNLI	Matched acc./Mismatched acc.	83.74/84.11	86.38/87.01	88.82/88.67
QNLI	Accuracy	90.32	92.02	92.77
RTE	Accuracy	64.98	64.98	72.56
WNLI	Accuracy	38.03	29.58	35.21

In order to find the best transformer that was applicable for all or most of the GLUE tasks, we tested using ELECTRA and several variations of BERT models including BERT, XLNet, roBERTa and DistilBERT as well as some of the size variations of each model (i.e. small, base, large). We found that electra-base-discriminator had a great performance on a majority of the tasks followed by bert-base-cased which was our benchmark and xlnet-base-cased, which

performed slightly better than BERT. As a result, we used these 3 models as a foundation of what we built our ensemble model on.

Ensemble

Before we began collecting the prediction data for our ensemble model, it was vital to understand how many models we use or we could risk getting even worse results than all the individual transformers. Therefore, we did an experiment on the first task, CoLA, to determine the accuracy scores of combining 1 through 6 models together for an ensemble. The results are shown in the graph below:



The transformer models we used for this graph were variations of ELECTRA, BERT and XLNet. In order to find the best results for this experiment, we decided to work backwards and start the ensemble at 6 models where the models that were repeated were the highest scoring models (ELECTRA and BERT). After that, we did several iterations of removing the lowest scoring model from the ensemble until there was one model left (ELECTRA).

By doing this we were able to assure that the accuracy of the ensemble was reasonable and the limit to how many models we should use to collect prediction data for the GLUE tasks. We can see from the graph that when the number of models equals 3, there is a peak followed by a dip when the ensemble quantity is 4. The transition from 5 to 6 models in the ensemble was a massive increase, so we decided to stick with 3 models in our ensemble. To put into further context, since this experiment was only for 1 task, we decided to look at the individual scores of the other GLUE tasks and XLNet and BERT demonstrated 2nd highest scores in different tasks which explains why our final ensemble includes ELECTRA, BERT *and* XLNet.

In order to create a viable ensemble model, it was necessary to output the predictions of our models of interest to feed them into an ensemble classifier. These predictions were based on the validation set of each of the GLUE tasks. The figure below shows how we formatted these

predictions into a new dataset. For each task, there is a target column indicating the actual target values in the validation set.

	electra-0	electra-1	xlnet-0	xlnet-1	bert-0	bert-1	target
0	-3.015364	2.818316	-2.792709	3.361839	-2.782302	3.292792	1
1	-3.272816	3.013487	-3.019989	3.219829	-2.536592	3.144536	1
2	-2.484975	2.380282	-1.700235	1.687799	-2.348618	2.650339	1
3	-3.138503	2.965038	-2.994277	3.023010	-2.814234	3.324904	1
4	-2.365048	2.197786	-2.657328	2.292092	2.783062	-3.252907	0

Due to the fact that most of the tasks contained binary or multi-class target variables, the ensemble classifier we used was Logistic Regression and Random Forest Classifier. After training and testing the predictions for each task on both these models, we found that the performance was extremely similar. STS-B was the only task that contained a continuous target variable, so we used Linear Regression as the basis of the ensemble model for that task. In order to test the performance of each ensemble, it was important we used the validation set as opposed to the test set.

The results of our ensemble model on each task is shown in the following table:

Task	Metric	Benchmark(Bert)	XLnet	Electra	B+X+E
CoLA	Matthews corr	59.55	46.76	67.74	74.04
SST-2	Accuracy	92.20	93.81	94.95	94.85
MRPC	F1/Accuracy	90.16/86.03	91.36/87.99	91.46/88.24	94.92/92.68
STS-B	Pearson/Spearman corr.	85.19/85.13	88.56/88.34	90.74/90.57	90.57
QQP	Accuracy/F1	90.79/87.58	90.99/87.93	91.79/89.06	92.41/90.11
MNLI	Matched acc./Mismatched acc	0.8882/0.8867	0.8673/0.8646	0.8397/0.8436	89.22
QNLI	Accuracy	90.44	92.02	93.03	94.00
RTE	Accuracy	56.68	66.43	81.23	83.92
WNLI	Accuracy	56.34	53.52	56.34	46.66

The results of our ensemble model were generally good in that none of the final metrics were below the lowest individual score of the 3 transformer models we used. Most of the final scores for the ensemble model were above or quite close to the highest scoring transformer model.

Summary and conclusions

Conclusion

Using the GLUE benchmark, we used transformers, state of the art language models, to fine-tune and get metrics for each task that consisted of single sentence prediction, similarity, paraphrase and inferencing. The primary model that we decided to focus on being ELECTRA, we tested out several other models that were mainly BERT framework based models and we ended up choosing BERT and XLNet as secondary models of interest.

Before training these models for predictions, we fine-tuned ELECTRA, BERT and XLNet by looking at the batch size, maximum sequence length, learning rate and number of epochs to get the best metrics possible. Since there were different metrics for each task, it was important to understand why the f1 score, Matthew's Correlation Coefficient, Pearson-Spearman Correlation Coefficient and accuracy score were being used.

Finally, we began the ensemble process by first generating predictions from our 3 transformers on the validation set for every GLUE task followed by creating a new dataset for each task where the target was the actual label of the validation set and the independent variables were the predictions. We compared logistic regression and random forest classifier for the binary and multi-class label GLUE tasks and found that the results were very similar. For the GLUE task that contained a continuous target, we used linear regression to ensemble. Our results were predominantly positive in that the ensemble score was never lower than the lowest individual transformer score and most times similar or higher than the best individual score. Ultimately, using transformers and the ensemble technique to complete the GLUE benchmark was extremely useful in our understanding of transformer models and allowed us to gain a deeper understanding of the ELECTRA, BERT and XLNet architecture and framework.

Future Improvements

After completing this project, we noted several things that we could have done better to achieve a higher score or certain techniques we could have implemented in order to make our ensemble more accurate. Although our current results demonstrated the usefulness and function of ELECTRA, BERT and XLNet, we believe that we could go even further in the future to produce better results.

In the future, we plan to test the large versions of each of the transformers we selected as well as others. In doing this, we can run glue tasks on transformers that were pre-trained on a much larger corpus and contain an increased number of parameters by sacrificing computation speed. By using transformers that perform well individually, the predictions that they make will contribute to a better ensemble model. Furthermore, we can fine-tune more types of parameters. We found that there is a mode_config.json file that is in each transformer file on huggingface.com which contains parameters such as embedding_size, attention_heads, hidden_layers, etc. If we spend more time tuning these parameters, we would develop a better understanding of how each transformer architecture works at each layer and possibly output better metrics. The reason we decided to focus on the main parameters we listed (sequence length, batch size, etc.) is because the paper written by the creators of these transformers mentioned that the parameters values in the model configuration produced the best results.

During hyper parameter tuning, the maximum sequence length is defined as the number of tokens within each row of the text column. To get the proper parameter value that is unique to each GLUE dataset, we plan to create a histogram of the length of each row in the text column.

From there, we can determine the average text length and initialize the proper max_seq_length parameter..

In order to create a better ensemble classifier, we thought it would be interesting to create an ensemble class that had each of our transformers of interest as layers. As opposed to running each model separately to collect the outputs, we could put the outputs of one model into the input of the next model by manipulating the number of neurons and hidden dimension sizes. As a result, the ensemble class could automatically output the predictions based on all 3 models combined. Another method that we considered was taking the average of the predictions we made from the 3 models and performing classification to train and test for accuracy. Finally, we would make predictions on the test set which we left unused in order to receive accuracy scores by uploading them to the leaderboard.

Reference

Huggingface. (n.d.). Transformers/examples/pytorch/text-classification at master · Huggingface/Transformers. GitHub. Retrieved December 9, 2021, from https://github.com/huggingface/transformers/tree/master/examples/pytorch/text-classification.

Huggingface. (n.d.). Transformers/examples/pytorch/text-classification at master · Huggingface/Transformers. GitHub. Retrieved December 9, 2021, from https://github.com/huggingface/transformers/tree/master/notebooks.

Clark, M. Luong, Q. V. Le, and C. D. Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. In ICLR.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. Advances in neural information processing systems, 32.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. arXiv preprint arXiv:1901.02860.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461.