Harish Ram

Individual Report

Introduction

The project that I worked on concerned the GLUE Benchmark which consisted of 9 tasks that required evaluating various natural language processing categories ranging from semantic analysis to inferencing.

My group decided to focus on transformer models for this project as they are very new models that have already demonstrated exceptional results on the GLUE leaderboard. ELECTRA, BERT and XLNet were the 3 main transformers that my project revolved around. In order to properly use these transformers, it was necessary to demonstrate a proper understanding of the actual architecture, fine-tuning the parameters as well creating an ensemble model that used the predictions from all the transformers to generate even better results.

Description of Individual Work

The work that I focused on for this project was becoming an expert on ELECTRA, understanding its architecture and using it for model evaluation. In order to properly understand ELECTRA, I read the paper to understand how the generator and discriminator were pre-trained jointly and how its computationally efficient nature could be useful in running our models. The way I decided on using ELECTRA was using code referenced by the huggingface Github repository to run the GLUE Benchmark tasks to get results. I saw that ELECTRA performed possibly the best out of all the transformers I used (XLNet, BERT, Distillbert and Roberta) for a majority of the tasks.

I also was responsible for understanding and fine-tuning the parameters such as batch-size, maximum sequence length, learning rate and number of epochs. In order to properly fine-tune these parameters, it was important that I understand what the parameters did; thus, from my research and testing, I understood that batch size corresponds to the number of samples that the model can input before updating the weights and biases of the model parameters. I determined that the best batch size for a majority of the tasks was 32 simply by running the models continuously. The learning rate was another parameter I focused on which is defined by the number of steps the model takes before updating the weights. This was tested in the same way and I surmised that a learning rate to the power of -3, -4, -6 and -7 did not increase our score and 2E-5 was the best learning rate.

Furthermore, I collected the predictions from our transformers and wrote the code for the ensemble model to output our final results. This was python code using Logistic Regression, Random Forest Classification and Linear Regression in which the final dataset had the independent variables as the predictions and the label of the validation set as the dependent variable.

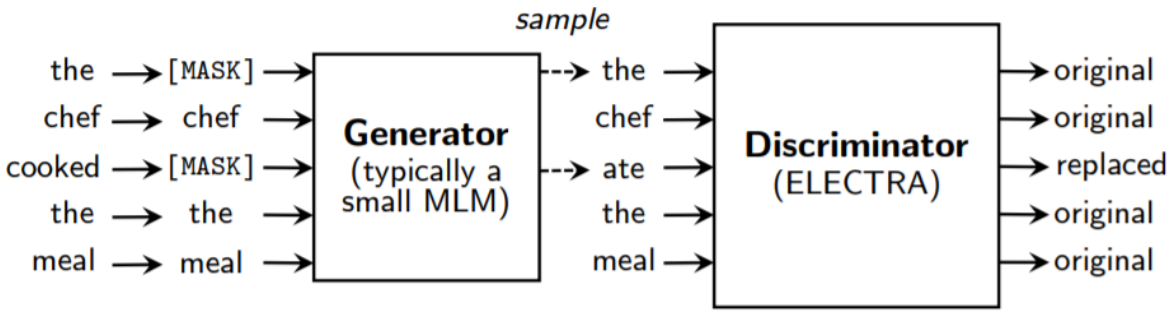
Harish Ram

electra-0	electra-1	xlnet-0	xlnet-1	bert-0	bert-1	target
-3.0153644	2.8183162	-2.7927089	3.3618393	-2.7823021	3.292792	1
-3.2728155	3.0134866	-3.0199888	3.219829	-2.5365918	3.1445363	1
-2.4849746	2.380282	-1.7002352	1.6877995	-2.3486183	2.6503386	1
-3.1385028	2.965038	-2.994277	3.0230095	-2.8142338	3.3249037	1
-2.3650482	2.1977863	-2.657328	2.2920923	2.7830622	-3.252907	0
-3.083772	2.889339	-2.8391738	2.7859573	-1.6859704	2.1696956	0
2.1689072	-2.1514554	-1.5155066	1.3749357	2.0328119	-2.2700455	0

After collecting this data, I inputted these datasets into the ensemble models and outputted the results based on the GLUE metrics.

Portion of Individual Work

The individual work I did started up with understanding the ELECTRA model in detail.



The primary reason why we chose our primary transformer of interest, ELECTRA, was because of its performance on the GLUE benchmark as well as its computational efficiency. ELECTRA uses an architecture that jointly pre-trains both a generator and discriminator. The generator is typically a Masked Language Model (MLM), but can be replaced with any type of model that produces an output distribution over tokens.

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

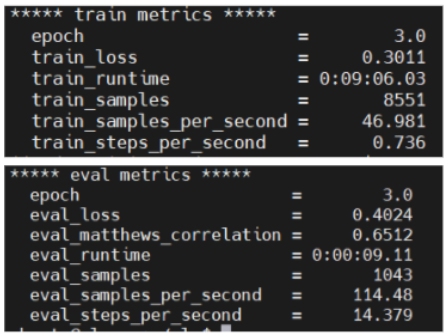
The function of the generator is to replace the masked tokens with plausible alternatives that the discriminator can learn from. The discriminator learns by iterating through every token in the corpus and using probability to determine whether the token was replaced by the generator or from the original corpus. The discriminator assigns a value to each token to determine this probability and passes it through a sigmoid output layer as shown above. This is known as Replaced Token Detection.

Harish Ram

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

The equations above reiterate how Replaced Token Detection works where the generator chooses the positions of where to replace the tokens, masks those tokens and replaces them with plausible alternatives. After pre-training, the model of ELECTRA is essentially the same as that of BERT. However, there are other features that ELECTRA is known for such as using a GAN (Generative Adversarial Network) that was manipulated to be trained on Maximum Likelihood as opposed to adversarially as well as the concept of weight sharing where the generator and discriminator share embedding weights on tokens generated and learned.

The next thing I did was perform parameter training on batch size, maximum sequence length, learning rate and number of epochs. My work looked similar to the table below for all the tasks and all the parameters in which I simply ran the GLUE Benchmark tasks by specifying the parameters and cross-referencing the training metrics with the evaluation metrics to finetune our models.

Doubling Batch Size - Reduces Run Time slightly -Decreases MatCorr Score slightly -More train samples per second - Less train steps per second	**google/electra-base-discriminator** **65.12**	python3 run_glue.py \--model_name_or_path google/electra-base-discriminator \--task_name cola \--do_train \--do_eval \--max_seq_length 128 \--**per_device_train_batch_size 64** \--learning_rate 2e-5 \--num_train_epochs 3 \--output_dir /tmp/cola/	

Finally, I wrote the ensemble code that used Logistic Regression, Random Forest Classification and Linear Regression to use the prediction results of ELECTRA, XLNet and BERT. The following figure shows how I compared the individual scores with the ensemble score. This code used the dataset shown above in which I inputted them into the ensemble models. I did this for all the tasks and compared them to the individual scores and found that our ensemble model was for the most part successful.

Harish Ram

Ensemble for MRPC (Microsoft Research Paraphrase Corpus) using:

Metric - F1/Accuracy

- ELECTRA (google/electra discriminator) - Metric: 0.9145/0.8823
- XLNet (xlnet-base-cased) - Metric: 0.9135/0.8799
- BERT (bert-base-cased) - Metric: 0.9015/0.8602

Ensemble Metric: 0.9268/0.9491

```
[ ]  #MRPC Classification Report using Logistic Regression
     lr_mrpc_pred, lr_mrpc_y_test = LR(mrpc)
     print(classification_report(lr_mrpc_y_test, lr_mrpc_pred))
```

```
                  precision    recall  f1-score   support

               0       0.95      0.80      0.87        25
               1       0.92      0.98      0.95        57

        accuracy                           0.93        82
       macro avg       0.94      0.89      0.91        82
    weighted avg       0.93      0.93      0.92        82
```

```
[ ]  #MRPC Accuracy/F1 Score using Logistic Regression Predictions
     lr_mrpc_acc = accuracy_score(lr_mrpc_y_test, lr_mrpc_pred)
     lr_mrpc_f1 = f1_score(lr_mrpc_y_test, lr_mrpc_pred)
     print(f"Using Logistic Regression Predictions: \n\nAccuracy Score: {lr_mrpc_acc} \nF1 Score: {lr_mrpc_f1}")
```

```
Using Logistic Regression Predictions:

Accuracy Score: 0.926829268292683
F1 Score: 0.9491525423728813
```

Results

For the ensemble model, we used the same GLUE Metrics as used in the benchmark.

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 figure above shows the results of the ensemble work that I did that involved using ELECTRA, XLNet and BERT, fine-tuning the parameters of each model as well as creating the ensemble to output these results. This table was created by a group member, but the ensemble scores were based on the code that I wrote. The table shows that the ensemble scores which were the same as the GLUE metric were never below the lowest-performing transformer and mostly higher than the best performing transformer with the exception of WNLI where the benchmark results were already low.

The scores of ELECTRA were the best in almost every category which is why we used it as part of our ensemble. XLNet also had very high metrics on individual GLUE tasks and BERT was our benchmark for this project which is why we included these 2 models in the ensemble.

Conclusion

Ultimately, there were still several things that we could've done better if we were to continue this project. The use of large models compared to base models would definitely be useful in getting higher benchmark results. For fine-tuning the maximum sequence length, it would also be important to do more analysis on the text length in each GLUE task to find the best length. When creating an ensemble, we would have ideally wanted to create a new model class in which the transformers that we used were layers in the model as opposed to getting predictions separately. Another solution we would like to have experimented on is to average our prediction results instead of putting them through a classifier. Finally, once we were able to make predictions on the test set, it would be important to actually submit our results to the GLUE leaderboard to get feedback on our results.

Code Percentage

67.71%

References

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.

Harish Ram