DATS 6203: NLP for Data Science

Individual Project Report

Arathi Nair
December 9, 2021

1. Introduction

This report comprises of a detailed description of the implementation of the transformer model, Electra - a self-supervised language transfer learning model used in this project. Our objective was to create an efficient framework that would provide an accurate output prediction for each task in the GLUE benchmark dataset. The dataset comprises of nine natural language understanding tasks, including of natural language inference, sentiment analysis, acceptability judgment, sentence similarity, and common-sense reasoning.

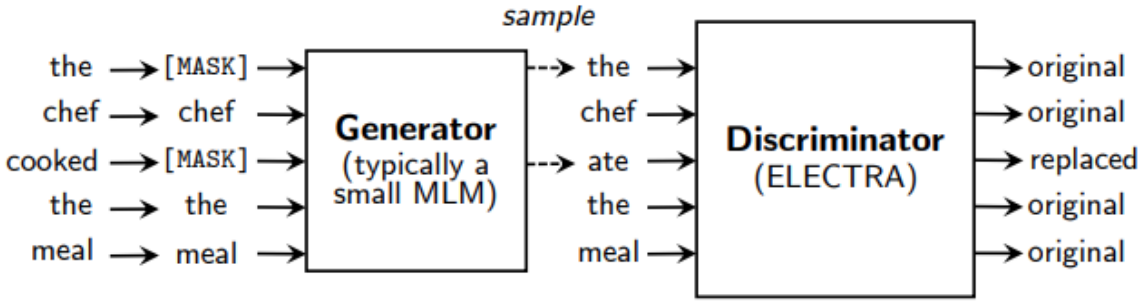
The transfer learning models that were chosen for this project models were ELECTRA, XLNET and DeBERTa. These models were initially trained to create a baseline performance on all the GLUE tasks; and to further improve our model's performance these models were combined to develop an ensemble model using Weighted Average Random Forest and Gradient Boosting on the trained models. Each team member trained and finetuned one of the models to create a baseline to input into our random forest and gradient boosting classifier model.

2. Background

The transformer model that was studied and implemented by me was the ELECTRA model. One of the disadvantages with using MLM pre-training is that it predicts only a small subset (15%) of the input token instead of predicting every single input token. This reduces the amount of information that can be learned from each sentence by the model. ELECTRA model architecture has a unique pre training approach called 'replaced token detection' for prediction unlike its predecessors that relied on MLM pre-training. The model can efficiently learn and classify token replacements accurately by including all input tokens.

The results obtained by the model can match or exceed the pretrained MLM model using relatively little compute. At small scale, ELECTRA achieves strong results even when trained on a single GPU. The contextual representations learned by the model can substantially outperform other MLM models given the same model size, data and computational resources. Considering all of these advantages, ELECTRA was chosen as the preferred model to analyze and evaluate the GLUE tasks.

Model Architecture



The ELECTRA framework consists of two networks a generator and a discriminator similar to a generative adversarial network (GAN). Like the GANs (generative adversarial networks) approach, the two networks are pit against each other. The generator is optimized to trick the discriminator with increasingly convincing "fake" data. The discriminator is then required to identify and separate the true inputs from the fake data. However, the generator of the ELECTRA model is not optimized to increase the loss of the discriminator but is instead trained like a typical MLM model where it must best guess the "[MASK]" values.

This approach to pre-training is more efficient than MLM because the model must consider every single token. This produces a model which has an improved comprehension of context. After pre-training is complete, the generator model is discarded, the new Electra transformer model can then be applied to handle various language processing tasks.

Pre-Training Process

First the encoder maps the sequence of the input token $x = [x_1, ..., x_n]$ into a sequence of contextualized vector $h(x) = [h_1, ..., h_n]$. To mask a token at a position (t) i.e., $x_t =$ "[MASK]", the generator outputs the probability of generating the x_t token using Softmax layer. The equation below represents the process of the generator.

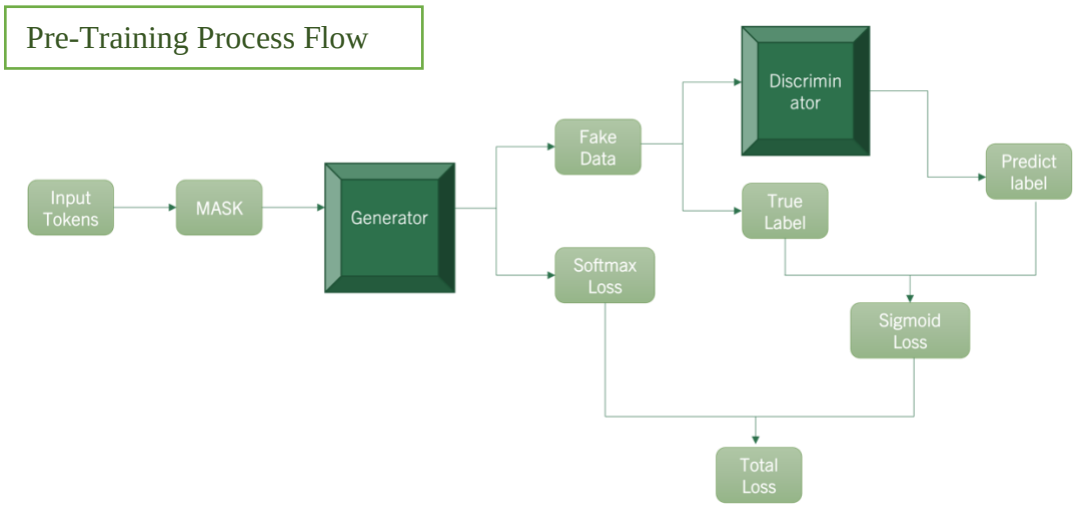
$$p_G(x_t|\boldsymbol{x}) = \exp\left(e(x_t)^T h_G(\boldsymbol{x})_t\right) / \sum_{x'} \exp\left(e(x')^T h_G(\boldsymbol{x})_t\right)$$

While for a given position (t), the discriminator predicts whether the token x_t is "real" or "replaced" using Sigmoid activation function layer. The equation represents the process of the discriminator.

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

Unlike GAN, the generator is trained using maximum likelihood rather than being trained to fool the discriminator. Finally, the combined loss of generator and discriminator is minimized before discarding the generator. The equation represents the calculation of the combined loss of the pre-training process.

$$\min_{\theta_G, \theta_D} \sum_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) + \lambda \mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D)$$



3. Model Implementation

The approach for training the GLUE dataset was fully reimplemented in PyTorch using **Electra-small-discriminator** model. Different libraries from Hugging Face, namely Transformers, Tokenizers and Datasets, have been used respectively for the model implementation & training loop logic, and for tokenization of the inputs.

In order to reduce computational time, we have implemented the Electra-Small model which consists of 12 layers, 256 hidden size and 14M parameters. As shown below, the model configuration class was set to default to train the tasks.

```
( vocab_size = 30522, embedding_size = 128, hidden_size = 256,
num_hidden_layers = 12, num_attention_heads = 4, intermediate_size =
1024, hidden_act = 'gelu', hidden_dropout_prob = 0.1,
attention_probs_dropout_prob = 0.1, max_position_embeddings = 512,
type_vocab_size = 2, initializer_range = 0.02, layer_norm_eps = 1e-
12, summary_type = 'first', summary_use_proj = True,
summary_activation = 'gelu', summary_last_dropout = 0.1, pad_token_id
= 0, position_embedding_type = 'absolute', use_cache = True,
classifier_dropout = None, **kwargs )
```

The preprocessing of the data was done by implementing pretrained class transformers**.ElectraModel** as the base model. The output below displays the model process for 1 Electra layer, the model follows the same BERT model except that it has a separation of the embedding size and the hidden size: the embedding size is generally smaller (128), while the hidden size (256) is larger. An additional linear layer is used to project the embeddings from their embedding size to the hidden size. The Electra Attention mechanism layer consists of three sub layers of query, key which is passed through Softmax to get attention probability to weight the impact of each token. The attention is then multiped by value to calculate the importance of each token to create.

```
(electra): ElectraModel(
  (embeddings): ElectraEmbeddings(
    (word_embeddings): Embedding(30522, 128, padding_idx=0)
    (position_embeddings): Embedding(512, 128)
    (token_type_embeddings): Embedding(2, 128)
    (LayerNorm): LayerNorm((128,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (embeddings_project): Linear(in_features=128, out_features=256, bias=True)
  (encoder): ElectraEncoder(
    (layer): ModuleList(
      (0): ElectraLayer(
        (attention): ElectraAttention(
          (self): ElectraSelfAttention(
            (query): Linear(in_features=256, out_features=256, bias=True)
            (key): Linear(in_features=256, out_features=256, bias=True)
            (value): Linear(in_features=256, out_features=256, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): ElectraSelfOutput(
            (dense): Linear(in_features=256, out_features=256, bias=True)
            (LayerNorm): LayerNorm((256,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): ElectraIntermediate(
          (dense): Linear(in_features=256, out_features=1024, bias=True)
        )
        (output): ElectraOutput(
          (dense): Linear(in_features=1024, out_features=256, bias=True)
          (LayerNorm): LayerNorm((256,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
```

The model architecture of the base Electra model is identical to the BERT model except that it adds an additional linear layer between the embedding later and the encoder if the hidden size and the embedding size are different.

To prepare the inputs for the model, class transformers.**ElectraTokenizer** with max length of the sequence of 512 was used to run an end to end tokenization of raw text and map their respective token ids. Class transformers.**DataCollatorWithPadding &** transformers**.DataCollatorForLanguageModeling** was used to build each batched set of the training dataset with dynamic padding to the maximum length of a batch. This tokenized dataset was then passed to the PyTorch **DataLoader** to process the inputs.

For sentence classification, class transformers.**ElectraForSequenceClassification** head was attached to the top of the pertained base model. This layer acts like a linear layer for sequence classification and regression on the pooled output from the base model.

```
(classifier): ElectraClassificationHead(
  (dense): Linear(in_features=256, out_features=256, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (out_proj): Linear(in_features=256, out_features=2, bias=True)
)
```

For optimization of the model, class transformers.**Trainer** was implemented to train and evaluate the neural network, calculate the loss function.

Finetuning

After obtaining a baseline score for all the GLUE tasks, the model was finetuned to achieve higher prediction scores and improve the performance of the model. For hyperparameter tuning, we experimented with different learning rates (5e-5, 2e-5, 1e-4), weight decay values (0.01and 0.1), number of epochs (3, 5,10) and batch sizes (8,10,12). The optimal results were obtained by setting the learning rate = 2 e-5, weight decay = 0.01, number of epochs = 3, batch size = 8 (except for WNLI task which was set as 10).

To further finetune the model parameters, we also implemented hyperparameter search using **Optuna** which is an automated optimization software that uses different machine learning algorithms such as grid search, random, Bayesian

However, the result achieved using **Optuna** did not yield higher scores compared to the previously finetuned model.

4. Results

The dataset was initially tested on the Electra-Small pretrained model with default parameter settings to obtain a baseline score on each GLUE task. Later to improve the model performance, the model parameters were finetuned by experimenting with different learning rates, batch sizes and weight decay rates along with the use of Optuna software. The best learning rate selected for all the tasks was 2 e-5 with weight decay of 0.01, number of epochs of 3, batch size of 8 (except for WNLI task which was set as 10). The results of the baseline and the finetuned scores are summarized in the table below.

Corpus	Metric	Baseline	Tuned
Single-Sentence Tasks			
CoLA	*Matthew's*	0.466	0.603
SST-2	*Accuracy*	0.883	0.917
Similarity and Paraphrase Tasks			
MRPC	*Accuracy and F1*	'accuracy': 0.742 'f1':0.805	'accuracy': 0.865 'f1': 0.903
QQP	*Accuracy and F1*	'accuracy': 0.879 'f1': 0.839	'accuracy': 0.898, 'f1': 0.863
STS-B	*Pearson and Spearman*	'pearson': 0.548	'pearson': 0.877, 'spearmanr': 0.875
Inference Tasks			
MNLI	*Accuracy*	0.799	0.820
QNLI	*Accuracy*	0.866	0.889
RTE	*Accuracy*	0.555	0.682
WNLI	*Accuracy*	0.436	0.603

Electra Median Expected Results From GitHub

	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE
Metrics	MCC	Acc	Acc	Spearman	Acc	Acc	Acc	Acc
ELECTRA-Large	69.1	96.9	90.8	92.6	92.4	90.9	95.0	88.0
ELECTRA-Base	67.7	95.1	89.5	91.2	91.5	88.8	93.2	82.7
ELECTRA-Small	57.0	91.2	88.0	87.5	89.0	81.3	88.4	66.7
ELECTRA-Small-OWT	56.8	88.3	87.4	86.8	88.3	78.9	87.9	68.5

The ELECTRA-Small transformer model was able to achieve significantly higher accuracy scores on all tasks compared to the baseline model and the ELECTRA median expected results for ELECTRA-Small model presented on the official GitHub page. For single sentence tasks, the model was able to obtain prediction accuracy score of 92% - SST2 and 61% - Cola. For similarity and paraphrase tasks, the obtained prediction F1 scores were 91% - MRPC, 90% - QQP and 88% - SSTB; and for inference tasks the obtained accuracy scores were 90% - QNLI, 82% - MNLI, 68% - RTE and 60% - WNLI.

5. Conclusion

From the above results, we observed that finetuning the initial baseline Electra-Small model drastically improved the performance. The model was able to obtain a high accuracy in classifying the labels and predicting the output for most Glue benchmark tasks.

Overall, we find that the model is most likely perform higher on textual similarity, rather than on an actual understanding of the text. During the evaluation of test set, it was noticed that the scores dropped from previous high scores, suggesting either overfitting on the training set, or that the test set has some domain shift that the transformer model couldn't generalize to. In the future, to improve the accuracy

scores even further, the GLUE datasets can be trained on a bigger Electra model like the Electra base or large models.

6. Code Percentage

Code from external source (Hugging Face): 95%	Code modified: 5%

7. References:

ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.
https://huggingface.co/docs/transformers/model_doc/electra