Enhanced generalizable minBERT model for multiple downstream tasks

Stanford CS224N Default Project

Yi Qi
Department of Computer Science
Stanford University
qiyi@stanford.edu

Abstract

This project aims to 1) build a BERT model that could work for Sentiment Analysis and extend the model to multiple downstream tasks 2) explore and analyze performance impact from different improvement approaches. For part 1, we've built a BERT model that could work for three downstream tasks and got 0.348 average score as baseline. For part 2, we explored a few different approaches and finally got 0.663 average score for test Set.

1 Key Information to include

- Mentor: Gabriel Poesia

2 Introduction

Text classification is a classic problem in Natural Language Processing (NLP). Pre-trained language models have shown to be useful in learning common language representations by utilizing a large amount of unlabeled data: e.g., BERT (Jacob Devlin and Toutanova (2018)).
In this project, we first implemented minBERT model using the base skeleton code from Github and it can get more than 50% accuracy for Sentiment Analysis. It produced a relatively good accuracy for single task but how about applying it to multiple tasks?
As a baseline implementation, we simply attached a linear projection layer for each of the prediction tasks on top of the BERT output and trained model using data from each target task one by one. As a result, it produced 0.383 accuracy (lower than running single task) for Sentiment Analysis and 0.622 for Paraphrase Detection. The result for Semantic Textual Similarity is only 0.040. The model worked poorly when handling three tasks together at the beginning, but could we make it better? We conducted several experiments regarding logit/loss calculation, training strategies and regularization to further improve the performance.

3 Related Work

How to Fine-Tune BERT for Text Classification? (Chi Sun and Huang (2019)) explained experiment results for several common training strategies to improve BERT model. Some experiments (e.g. further pre-training) is also explored in our project.

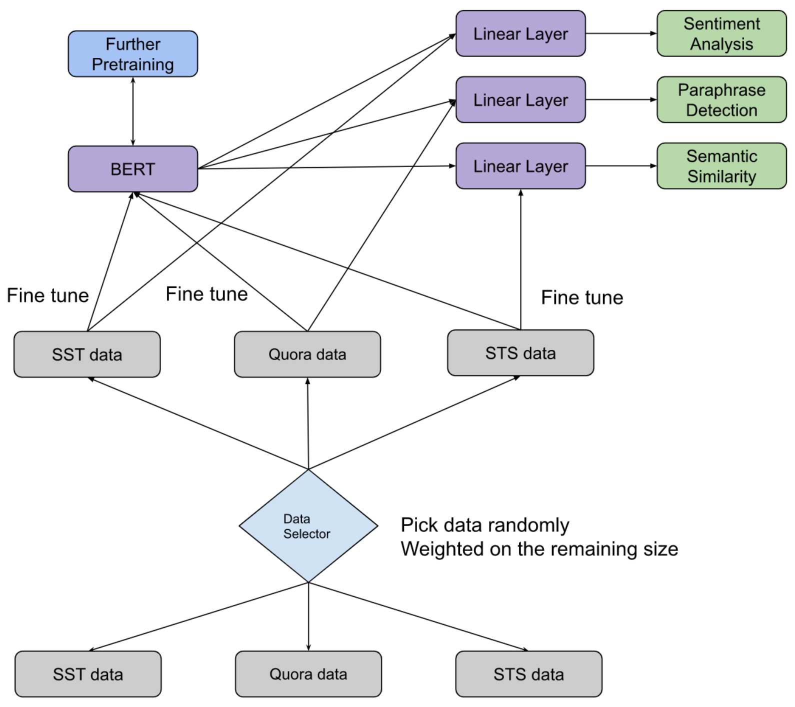
4 Approach

4.1 Architecture

It uses a similar architecture as mentioned in the handout.
For sentiment analysis, we passed the BERT output to a dropout layer and linear transformation layer.

The output was then passed to a softmax function to get possibility distribution. We used argmax to get the prediction.
To consume two embedding inputs for paraphrase detection, we first computed the absolute value of the difference of the two embeddings and passed it to a dropout layer and linear layer. We rounded the sigmoid value to get the predicted label.
To consume two embedding inputs for semantic textual similarity, we passed the BERT output to a dropout layer and linear layer, and then compute cosine similarity and scale it to the correct range.

Besides that, an extra pre-training layers (Use masked language model) are added on top of BERT model. To fine tune three downstream tasks fairly, we added a data selector to select batches from different datasets randomly based on remaining size of each data size.



4.2 Baseline

The baseline is the initial result for multi-task classifier without adding extra improvements.
It simply added dropout layer and linear projection to each of three tasks on top of BERT embedding. We used cross entropy for Sentiment Analysis, and mean square loss for the other two. In each epoch, we trained the entire data set from each task one by one.
As a result, it got 0.348 average score.

5 Experiments

5.1 Data

We used the dataset provided from the default project, you can see details from the handout
For Single task Classifier (Sentiment Analysis), we used Stanford Sentiment Treebank (SST) dataset and CFIMDB dataset
For Multi-task Classifier: We used Stanford Sentiment Treebank for Sentiment Analysis, Quora Dataset for Paraphrase Detection, and SemEval STS Benchmark Dataset for Semantic Textual

Similarity.

5.2 Evaluation method

For Sentiment Analysis, we compute the predicted label and evaluate the prediction accuracy by comparing with the true labels.
For Paraphrase Detection, we compute the predicted label and measure the prediction accuracy by comparing with the true labels.
For Semantic Textual Similarity, we compute the predicted rating and measure the Pearson correlation between our prediction and true labels.

5.3 Experimental details

We first implemented the minBert model and used it to do Sentiment Analysis and used default options for the training: 10 Epochs, learning rate: 1e-3 (pretrain mode), 1e-5 (fine tune mode), dropout rate: 0.3 and Batch size = 64.

After we got a decent score (0.510 for SST fine tune mode), we extended it to multi-task prediction. As a base line, we simply added dropout layer and linear projection to each of three tasks on top of BERT embedding and used cross entropy for Sentiment Analysis, MSE for Paraphrase Detection and Semantic Textual Similarity. In each epoch, we trained the entire data set from each task one by one.

Then we started with exploring different approaches to improve the performance, we started with the default options for the training: 10 Epochs, learning rate: 1e-5, dropout rate: 0.3 and Batch size = 64. We adjusted the options while we did the experiments.

- **Train strategy**. To run data from multiple datasets, instead of running each dataset one by one, we chose batch from one of the three tasks randomly weighted on the remaining size of the dataset. We don't want to do a fully random selection since the sizes of different datasets are very imbalanced.
- **Logit for similarity**. We explored two types of logits to represent similarity for paraphrase detection and semantic textual similarity tasks. One is to use difference of the two input embeddings and the other one is to use cosine similarity.
- **Loss function**. We changed the loss function from mean square error to binary cross entropy for Paraphrase detection task.
- **Regularization**. We found the training accuracy was much higher than the dev accuracy which indicated an overfitting problem. So we increased weight decay and dropout rate. We tried a few weight decay: 0.1, 0.5, 1.0, 10 and tried dropout rate 0.45, 0.5, 0.55, 0.6.
- **Embedding**. We tried to use average of the last layer of the BERT model's output instead of using the pooler output for embedding. We explored two options: average on all words and only average on the actual words (excluding the paddings).
- **Further pretrain**. We planned to build an extra pre-training layers (Use masked language model) to train the embedding using task domain data, but due to time constraints, we don't complete the implementation before submission time. We will complete it afterwards.

5.4 Results

Experiment Results (Single task, Dev Set)

Experiment	Accuracy
SST pretrain	0.348
SST finetune	0.510
CFIMDB pretrain	0.776
CFIMDB finetune dev accuracy	0.967

Experiment Results (Multiple task, Dev Set)

Experiment	AVG	SST	Quora	STS
Baseline	0.348	0.383	0.622	0.040
Single task classifier	-	0.510	-	-
Cosine Sim for STS	0.487	0.512	0.492	0.458
Diff Embedding for Quora	0.490	0.502	0.549	0.420
Binary Cross Entropy for Quora	0.616	0.509	0.787	0.552
Increase Weight Decay	0.624	0.524	0.789	0.558
Average BERT Embedding	0.645	0.504	0.805	0.626
Increase Dropout Rate	0.670	0.508	0.792	0.711
Further Pretrain	?	?	?	?

Experiment Results (Multiple task, Test Set)

Experiment	Accuracy
SST test Accuracy	0.514
Paraphrase test Accuracy	0.791
STS test Correlation	0.683
Overall test score	0.663

SST: Stanford Sentiment Treebank (Sentiment Analysis) .
CFIMDB: Sentiment Analysis
Quora: Paraphrase Detection.
STS:SemEval STS Benchmark (Semantic Textual Similarity).

6 Analysis

- **Train strategy**. At the beginning, We were simply running batch from each dataset one by one and it gave us a poor (0.348) average score and the score for Sentiment Analysis is only 0.383, which is much lower than the score (0.510) when we run single sentiment analysis task. It is kind of expected since if we run dataset one by one, the last dataset will have unfair advantage and rune the embedding fine-tuned from the other datasets. After we did the sample selection based on the remaining size of each dataset, it provided a more smooth data consuming and increased the sentiment analysis score to a similar score as running sentiment analysis separately.
- **Logit for similarity**. Using diff of two input embeddings works better than cosine similarity for paraphrase detection; but cosine similarity works much better for semantic textual similarity. My understanding is when the two sentences are very similar, using diff of embeddings could capture whether or not the two sentences are similar very well (This is essentially what is needed for paraphrase detection). But when the two embeddings are not very similar and then we need to measure the degree of similarity; the difference of embeddings approach will lose too much information and cannot provide fine grained similarity differences.
- **Loss function**. We changed the loss function to binary cross entropy for Paraphrase detection task from MSE. This improved the performance of Paraphrase detection by around 20% accuracy. Mean square loss assume that the underlying data has been generated from a normal distribution while in reality, a dataset that can be classified into two categories is from a Bernoulli distribution.
- **Regularization**. We found the training accuracy was much higher than the dev accuracy which indicated an overfitting problem. So we increased weight decay and dropout rate. We explored different values of the "weight decay" but it only helped a little. We then increased the dropout rate to 0.5; it did give us a 3% accuracy increase. After we further increased the dropout rate to 0.6, we cannot get a better result. So we think 0.5 dropout rate is probably a good balance point.
- **Embedding**. We tried to use average of last layer of the BERT model instead of using the pooler output. It did increased the performance by 2% average accuracy. It probably because using the entire layer could preserve more information of the original sentences. We also further explored only average on the actual words (excluding the padding). It turned out that

averaging on all words gave better results than excluding padding words, which is kind of unexpected; I guess the reason might be because the padding output is already part of the BERT training and it could somewhat carry extra information (e.g. sentence length).

7 Conclusion

We could achieve a decent test score (avg: 0.663) by sharing the same BERT embedding layer among the three downstream tasks. Using correct loss function for classification problem (Paraphrase Detection) is very important for the classification performance. Using average of the last layer of the BERT embedding's output also gave good improvement compared to use the pooler output. It's fairly easy to get overfitting in the BERT based NLP system and a proper regularization will also provide a good improvement on the overall performance.

References

Yige Xu Chi Sun, Xipeng Qiu and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *arXiv:1905.05583v3*.

Kenton Lee Jacob Devlin, Ming-Wei Chang and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*.