Fine-tuning minBERT For Multi-task Classification

Stanford CS224N Default Project

Jimmy Ogada
Department of Computer Science
Stanford University
jimogada@stanford.edu

Abstract

Pretrained language models like BERT provide a good foundation on which to build task-specific models due to having feature-rich, extensible embeddings that can perform well on a wide variety of tasks. I propose a BERT implementation that enriches embeddings by fine-tuning them on in-domain data, for use in multi-task classification across three downstream tasks: sentiment analysis, paraphrase detection and semantic text similarity. Empirically, I observed negative interference in the tasks of sentiment analysis and semantic text similarity during training, likely due to contrasting optimal solutions; finetuning BERT on the three tasks simultaneously in a round-robin fashion resulted in low accuracy for the sentiment analysis task. By additionally finetuning BERT embeddings on just the sentiment analysis dataset, the overall accuracy increased across all three tasks. The final model outperformed the baseline with an overall score of 0.738, with strong performance on the individual tasks: achieving 0.533 on the SST test set, 0.82 on paraphrase detection test set and 0.726 on the semantic text similarity test set.

1 Key Information

- Mentor: Yuan Gao
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

Pre-trained language models like BERT offer several advantages over baseline models trained from scratch using contextual embeddings. BERT is a multi-layer bidirectional transformer trained on the tasks of masked word prediction and next sentence prediction. BERT's training on large, unstructured, heterogenous text, allows it to learn feature-rich representations that encode useful general knowledge like grammatical rules, logical reasoning, multilingual information, e.t.c., making BERT embeddings desirable for use across different natural language tasks. Prior to pre-trained models like BERT, models would be trained from scratch and have to learn such general knowledge information during training, leading to high compute costs and decreasing the model's ability to generalize across different tasks.

Although BERT offers good performance from pre-training and can handle a wide variety of tasks, multitask classification still poses a significant challenge: multi-task classification models can fail to capture nuances due to having more generic embeddings. Further, downstream tasks can interfere with each other during training due to conflicting objectives, leading to a model with lower accuracy. If two tasks have different optimal solutions in the embedding space, updating the embeddings for one task may negatively impact performance for the other task.

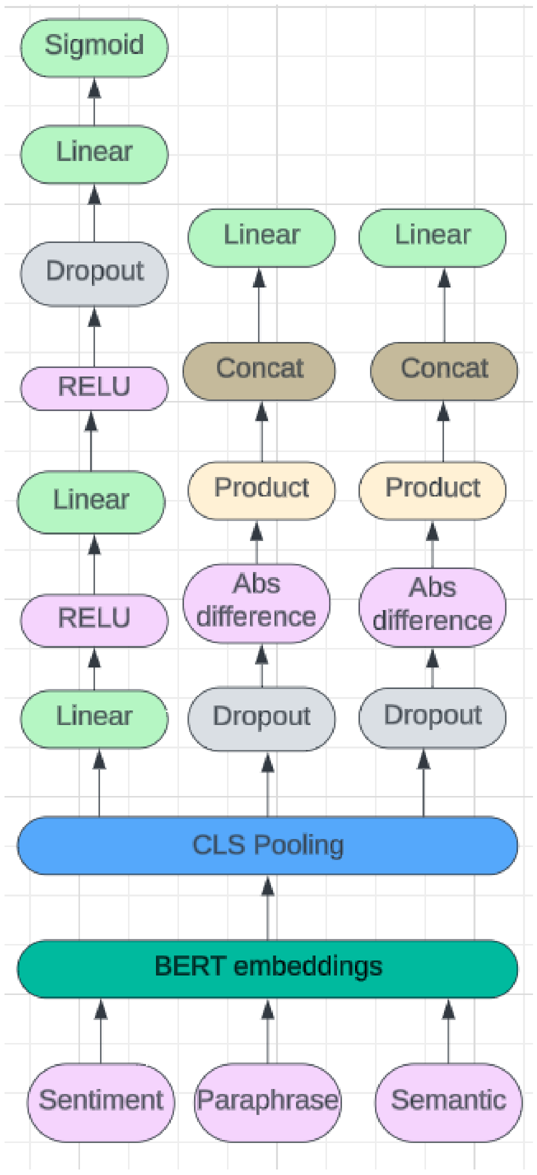

Figure 1: Model architecture

3 Related Work

Previous work from Sun et al (2019) has shown that base BERT's performance can be improved by finetuning BERT in three different ways: further pre-training BERT on in-domian data, finetuning BERT on mutlitask classification for related tasks, or finetuning BERT on the target task. Sun et al proposed adding a classification layer on top of the BERT model and then fine-tuning it on a labeled dataset for the target task. This approach motivated the work in this paper since finetuning is a relatively straightfoward approach that yields good performance without major model infrastructural changes. Further, the three text classification tasks of sentiment analysis, paraphrase detection and semantic text similarity are closely related so there is reason to believe that they can benefit from sharing the same BERT weights and finetuning them on their domain data. The simplicity of the additional finetuning paradigm is a key motivation for preferring it as the final approach.

4 Approach

I present the general architecture of the model in this section: In the model, I encode input tokens in the forward pass using the pretrained BERT model and then extract BERT embeddings from the CLS pooled token without applying dropout, in order to maintain flexibility in the level of regularization I use across the three different tasks. For sentiment classification, I apply two levels of linear transformations along with RELU activation on the pooled BERT token before applying dropout to prevent over-fitting. After applying dropout, I use a final linear layer to produce the final logits and then finally compute cross entropy loss over the training examples. Following this approach yielded a final accuracy of 0.533 for the SST task.

For Paraphrase Detection, I pass the batch of sentences through the forward pass to produce initial BERT embeddings, then apply dropout on the resulting tokens. I then take element-wise absolute difference to capture the dissimilarity information. I also an perform element-wise product of the pooled tokens to capture the similarity of the two tokens. If the two tokens are similar, they will have a small absolute absolute difference, and a large element-wise product. After, I concatenate the resulting tokens from the absolute difference and element-wise product operations into a single tensor that serves as a unified representation that contains similarity and difference information for the pairs of sentences. Finally, I pass the resulting tensor through a linear classifier to produce the classification logits and then compute binary cross entropy loss over the training examples.

For Semantic Textual Similarity, I pass the pairs of sentences through the forward pass to generate initial embeddings. I then compute the absolute difference of the resulting tokens, take an element-wise product, as in the paraphrase detection task, and concatenate the two resulting embeddings. Afterwards I apply a linear classifier to produce a single logit. Finally, I using a sigmoid activation function to scale the final logits to be between 0 and 5.

5 Experiments

5.1 Data

Stanford Sentiment Treebank (SST) dataset
I used the Stanford Sentiment Treebank which contains 11,855 single sentences extracted from movie reviews. The dataset was parsed with the Stanford parser2 and includes a total of 215,154 unique phrases, annotated by 3 human judges. Each phrase has a label of **negative**, **somewhat negative**, **neutral**, **somewhat positive**, or **positive**.

Below are some examples from the SST dataset:
Movie Review: Light, silly, photographed with colour and depth, and rather a good time.
Sentiment: **4** (Positive)

Movie Review: ... a sour little movie at its core; an exploration of the emptiness that underlay the relentless gaiety of the 1920's ... The film's ending has a "What was it all for?"
Sentiment: **0** (Negative)

Quora database
The website Quora3 often receives questions that are duplicates of other questions. To better redirect users and prevent unnecessary work, Quora released a dataset that labels whether different questions are paraphrases of each other.

Below are examples extracted from the Quora dataset:
Question Pair: (1) "What is the step by step guide to invest in share market in india?", (2) "What is the step by step guide to invest in share market?"
Is Paraphrase: **No**
Question Pair: (1) "I am a Capricorn Sun Cap moon and cap rising...what does that say about me?", (2) "I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What does this say about me? Is Paraphrase: **Yes**

SemEval STS Benchmark (STS) dataset
The semantic textual similarity (STS) task seeks to capture the notion that some texts are more similar than others; STS seeks to measure the degree of semantic equivalence [4]. STS differs from paraphrasing in that it is not a yes or no decision. Rather, STS allows for degrees of similarity.

5.2 Evaluation method

I evaluated the model on perform in the following way:
Sentiment analysis: **accuracy**
Paraphrase Detection: **accuracy**
Semantic Text Similarity: **Pearson correlation score**

5.3 Experimental details

I pre-trained the first model with a learning rate of 1e-3 over 10 epochs. Pretraining the model took around 2 hours in total. For the baseline model, I implemented a single layer classifier for the Sentiment Analysis task, token difference and element-wise product for Paraphrase Detection, and Cosine similarity for the Semantic Text Similarity tasks. Cosine similarity seemed like a fairly straightforward and promising approach, but after the first run, I found outputting Cosine scores as the output to the classifier led to consistently reduced accuracy for the Semantic Text Similarity task. The average accuracy using Cosine Similarity hovered around 0.016, often accumulating negative loss. After switching to the architectural approach I had implemented in paraphrase detection, I observed that the accuracy for the STS task improved significantly: accuracy went up from 0.016 to 0.376. The Paraphrase Detection task showed good performance from the start and remained consistent throughout all the runs. A possible explanation for this is that the Paraphrase Detection task used the largest dataset of the three tasks, so the BERT embeddings were adequately finetuned for the task. Further, I used a much more complex algorithmic approach in Paraphrase Detection: adding element difference and product as additional input features to the model most likely enhanced learning. However, despite decent average performance, the pre-trained model did not show enough promise, capping at 0.536 before additional architectural changes.

Thereafter I shifted my approach to finetuning the BERT model and saw the accuracy go up much higher than before. The overall accuracy improved from 0.536 to 0.653. I finetuned the BERT model using a learning rate of 1e-5 over 10 epochs for the three tasks in a round-robin fashion, propagating the loss after each training. Notably, the accuracy for the Sentiment Analysis task dropped significantly from 0.396 to about 0.24, prompting an investigation into the cause for the reduced accuracy. At this point, I was still using a single linear classifier in the Sentiment Analysis task so I added two linear layers with RELU activation to see if that could improve the low scores. The accuracy for the SST task slightly improved from 0.240 to 0.284, but it still remained below the previous levels of the pre-trained model. Worryingly, the paraphrase detection and semantic text analysis tasks retained robust performance. They each had accuracies of above 0.8 after fine-tuning the model. I investigated the cause by freezing the STS and Paraphase Detection tasks, and finetuning the model on just the SST dataset for 7 more epochs. Doing this restored the SST accuracy to 0.516, which was higher than had been seen so far. key drawback of the finetuning approach was that it would take a very long time to train (at least 12 hours for a full run), significantly limiting ability to perform iterative experiments.

5.4 Results

Model A: Pre-trained + Cosine STS
Model B: Pre-trained + Linear Layer STS + No Cosine STS
Model C: Fine-tuned with round-robin
Model D: Fine-tune with round-robin + 2 More Linear Layers in SST
Model E: Fine-tune with round-robin + 10 Additional SST epochs
Model F: Fine-tune with round-robin + 10 Additional SST epochs + 3 Additional STS epochs

Model	Overall score	SST score	Para score	STS score
Model A	0.428	0.364	0.729	0.016
Model B	0.536	0.396	0.524	0.376
Model C	0.653	0.240	0.810	0.818
Model D	0.664	0.284	0.800	0.815
Model E	0.717	0.516	0.828	0.615
Model F	0.738	0.533	0.82	0.726

Table 1: Comparison of model variations.

- Final results from the test leaderboard were as follows:
  Overall score: 0.738
  Sentiment Classification Accuracy: 0.533
  Paraphrase Detection Accuracy: 0.82
  Semantic Textual Similarity Accuracy: 0.726.

- Final results in the dev set leaderboard were as follows:
  Overall score: 0.738
  Sentiment Classification Accuracy: 0.526
  Paraphrase Detection Accuracy: 0.823
  Semantic Textual Similarity Accuracy: 0.730

  These results were much better than expected since the model used a fairly straightforward and simple approach to increase accuracy: performing additional fine-tuning on the BERT model and slightly correcting task interference by focusing finetuning on the task with decreased accuracy. Accuracy was highest for paraphrase detection which would be expected since it is trained on the Quora dataset which has the most examples. The large dataset likely had inordinate influence on embeddings. The sentiment analysis task experienced the lowest accuracy among the three tasks, which was slightly unexpected, especially after adding two additional layers with activation to enable the model to learn more complex and nuanced representations. It was surprising to discover that Cosine similarity performed the worst on accuracy when used in the semantic text similarity task. It is likely that the logits from the Cosine similarity were squished together, preventing the model from learning nuanced representations. This might explain why the newer model performed better with the added features of absolute difference and element-wise product.

6 Analysis

The finetuned BERT multi-task model shows robust performance across the three different tasks of sentiment analysis, paraphrase detection and semantic text similarity. This paper affirms that by combining the feature-rich embeddings of pretrained language models like BERT with additional finetuning on in-domain data, one can build powerful models that can perform well across a wide variety of tasks. Using BERT's pretrained embeddings as initialization for a model seems like the most prudent approach if one wishes to bootstrap a model for a target task.

A potential drawback of the implementation of this model is that by using very similar architecture for paraphrase detection and semantic text similarity, the two tasks may have become too coupled and impacted the weights of the model too strongly, leading to reduced accuracy for the other task that uses the model. This could be a factor in the low accuracy of the semantic analysis task after simultaneous finetuning on the three tasks, although the reason for this is inconclusive. A possible solution might be to standardize the architecture across the three tasks as much as possible to reduce any disparate impact of architectural choices affecting model performance. The Sentiment Analaysis and Sementic Text Similarity tasks were also trained on much smaller datasets so it is possible the model did not train robustly. Increasing the size of the datasets used to train the tasks sharing one model would ensure that the model weights are balanced and can generalize well across different tasks.

7 Conclusion

I presented a BERT implementation which uses additional finetuning to enhance embeddings for use in the three tasks of sentiment analysis, paraphrase detection and semantic text similarity. The model achieves robust performance across each of the three tasks, obtaining an overall accuracy of **0.738** with a simple architecture that is easy to implement and evolve. This means that the model can potentially achieve even much better performance if additional optimizations are layered on top on top of the finetuned embeddings. The results demonstrate the power of pretrained language models with additional finetuning on in-domain tasks.

References

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