Not-So-SMART BERT

Stanford CS224N Default Project

Eliot Jones
Department of Computer Science
Stanford University
eliot.jones@cs.stanford.edu

Abstract

Since their creation, $BERT$ models have become the state-of-the-art for multiple NLU tasks. One of the most common methods of evaluating BERT models is the General Language Understanding Evaluation (GLUE) benchmark, from which we have been tasked with utilizing and extending the $BERT_{BASE}$ model to achieve high performance on three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. This project performed experiments on multiple configurations of models, and determined that achieving high performance on the test leaderboard was best achieved with a simple, larger model.

1 Key Information to include

- Mentor: Soumya Chatterjee

2 Introduction

When the $BERT$ (Devlin et al., 2019) model was first introduced as a method for solving natural language understanding (NLU) tasks, it already outperformed then-current state-of-the-art models. In the years since, there have been multiple enhancements to the $BERT_{BASE}$ model, which have made further improvements upon the original model's performance on NLU benchmarks.

The task at hand was to implement our own version of $BERT_{BASE}$, and then adapt it to three different NLU methods, which are a part of the GLUE benchmark. Those three tasks were sentiment analysis using the Stanford Sentiment Treebank (SST) (Socher et al., 2013), paraphrase detection with Quora Question Pairs (QQP) (Fernando and Stevenson, 2009), and semantic textual similarity with the SEMEval STS dataset (Agirre et al., 2013). Each of these tasks have their own separate challenges. Sentiment analysis requires the model to be able to tell between positive, negative or neutral tones; paraphrase detection requires the ability to determine what groups of words convey the same semantic meaning; and semantic textual similarity broadens the scope of textual similarity, allowing for a range of five different determinations, from "not at all related" to "same meaning."

The above series of tasks present a challenge for the $BERT_{BASE}$ model, judging by baseline performance. For example, the initial implementation of $BERT$, which was evaluated on the SST dataset, as well as a separate movie review dataset, called CFIMDB. As outlined in section 5, baseline versions of $BERT$ found it quite difficult to accurately classify data from the SST dataset. As a result, in the following parts of the project, we were required to make improvements to the baseline model, in order to rank higher on a leaderboard.

Success for this project was determined based on the aforementioned leaderboard rankings. Plainly, achieving a higher score on all three of the tasks resulted in a greater success than the alternative. As a result, we had to be extremely results-oriented in our approach. Yet another factor was training time, with finetuning more complex enhanced $BERT$ models taking up hours at a time. In order to combat compute and temporal restraints, at first our idea was to modify the architecture of the $BERT$ model, in order to utilize a more lightweight version during finetuning. However, it quickly became clear that this method would not be feasible, since, as a baseline, reducing model size results in a drop in accuracy.

After this revelation, our methods and approach (highlighted further in section 4), shifted to focus completely on enhancing performance on the three datasets: SST, QQP, and STS. In the end, we determined that a three-headed model, built on top of the 'bert-base-uncased' version of $BERT_{BASE}$, with two intermediate layers, resulted in peak performance, out of all of our experiments.

3 Related Work

Projects related to $BERT$ would not be possible without Devlin et al. (2019)'s implementation of the $BERT_{BASE}$ model. However, the current state-of-the-art results on the GLUE benchmark (and by extension the SST, QQP, and STS datasets), are based upon much more than just a baseline model. As a result, the overall goal of our implementation was to identify key areas of weakness during each iteration, and determine ways to iteratively improve our model. As such, we took inspiration from the implementations outlined in this section.

3.1 Cosine Similarity Finetuning

In their paper "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," Reimers and Gurevych (2019) suggest a method to compare the semantic similarity between sentences during finetuning. Specifically, they discuss three different objective functions when it comes to predicting semantic similarity: a classification objective function, a regression objective function, and a triplet objective function, all of which are used with computed cosine similarities between sentences, after utilizing a pooling operation on top of the outputs of the $BERT$ models. With respect to our implementation, this method was not introduced directly as a result of seeking targeted improvements. However, it is an implementation which provided improvements upon the state-of-the-art at the time of its publication, and made sense to include in our model as well.

3.2 Smoothness-Inducing Regularized Optimization

A quick evaluation of the current state-of-the-art techniques for achieving high performance on the QQP and STS datasets yields the observation that Smoothness-Inducing Regularized Optimization (SMART) (Jiang et al., 2020) is a critical part of any model which hopes to attain top scores. The proposed method is a way to control model complexity during fine-tuning, which is an issue for more advanced architectures. In order to do so, the authors introduce small perturbations into the embeddings, and penalize large divergences or differences in model output, using either KL-divergence or squared loss depending on task type (classification or regression). However, one drawback with this method is extended training time and more computing resources, which has been discussed as an important factor during this project.

3.3 Heinsen Routing

A similar examination of state-of-the-art techniques on the SST dataset leads to "An Algorithm for Routing Vectors in Sequences" by Heinsen (2022). This algorithm attains number one status on the SST 5 fine-grained classification task, which as seen earlier proves to be quite difficult for $BERT_{BASE}$. As a result, during our experimentation, it was in our best interest to attempt an implementation utilizing Heinsen Routing, in order to improve model performance. The Heinsen Routing algorithm is simply a modified version of expectation maximization, which when combined with an updated version of $BERT$, $RoBERTa$, achieves peak accuracy on the task at hand. Similarly to SMART, this method also introduces more model complexity and extended runtime.

4 Approach

In order to better contextualize the methods used in section 5 of this paper, we will first outline the baseline model used for all experiments, and then discuss the different model architectures that were utilized during the finetuning process. For the sake of simplicity, we will denote the three-headed model built on top of $BERT_{BASE}$ as $BERT_{Multi}$, in reference to the multitask classification task it has been created for. Please refer to Devlin et al. (2019) for information on baselines outside of our own experimental ones.

4.1 Initial Model

Our initial task was to implement our own version of the $BERT_{BASE}$ model architecture as described by Devlin et al. (2019), and according to the skeleton code provided in the project description. Our initial implementation of the BERT model utilizes embedding layers, 12 BERT encoding layers, and a pooling layer, in order to use the pretrained $BERT_{BASE}$ embeddings to pass the sanity check.

Then, we implement the AdamW optimizer according to the algorithm provided in the project description, as well as the "efficient" implementation used by Kingma and Ba (2017). Once these methods have been completed, we run two instances of our basic BERT model, either pretraining with frozen parameters or finetuning them, and compare our results to the provided baselines for the SST and CFIMDB datasets. We then move on to implement the $BERT_{Multi}$ Model.

4.2 $BERT_{Multi}$ Model

The $BERT_{Multi}$ model is based upon the $BERT_{BASE}$ model, with an additional three heads, each of which modifies the outputs from $BERT_{BASE}$ in order to better align with the tasks at hand. The general model flow is outlined in Figure 1. In addition to the classical $BERT$ architecture, the $BERT_{Multi}$ model contains an additional intermediate layer, which is then followed up by a classification layer, which outputs the logits required for the three tasks. Specifically, each intermediate layer is formed by a series of mini-layers, with each mini-layer comprised of a linear activation, a GELU function, and then a dropout layer. Once the embeddings have passed through the intermediate layers, they are passed through a final linear layer, which outputs the logits.

For the sentiment prediction, since we only have one pair of a set of input ids and an attention mask, in order to get predictions we run these through the $BERT_{BASE}$ model to get embeddings, then through the intermediate layers, and then finally through the linear classifier. However, for both paraphrase detection and semantic similarity, we are provided with two pairs of input ids and attention masks, which results in a changed approach. For paraphrase detection, we first get the embeddings of both sets of tokens corresponding to the two sentences in question, and take the absolute value of the difference between these embeddings. After this has been calculated, we proceed as with sentiment prediction. Similarly, for semantic similarity, we obtain the embeddings like we did with paraphrase detection, normalize the two sets of embeddings, and then compute the dot product between them. This process is identical to calculating the cosine similarity, as outlined by Reimers and Gurevych (2019). These similarity scores are then passed through the intermediate and classification layers.

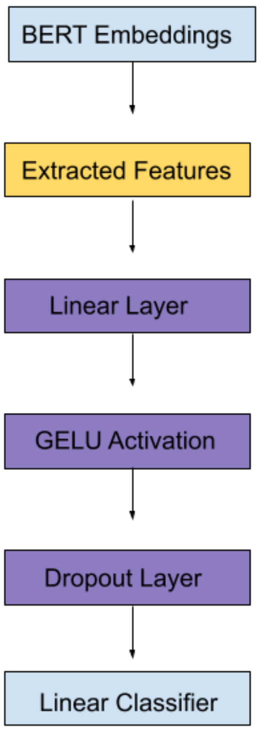
4.3 Finetuning

Since completing any pretraining using our model doesn't update weights in any fashion, we made the decision to exclusively perform finetuning using our model, in order to give ourselves the most possible chances at achieving a high-performing model. Figure 2 describes the flow of our finetuning loop, which utilized a multitask approach. Specifically, as visible in the figure, we perform multitask classification, updating parameters based on the gradients accumulated across the three tasks. We finetune on the selected batch of SST data, then QQP data, and then finally STS data, backpropagating the task-specific loss each time before continuing to the next. For the SST data, we utilize pytorch's cross entropy as our loss function, for QQP the binary cross entropy with logits function, and for STS we utilize mean squared error loss, which aligns exactly with Reimers and Gurevych (2019)'s regression objective function. It is also during this loop that we investigated the integration of SMART regularized optimization, in which we calculated perturbations of the embeddings from a given batch sample and appended the loss to the original loss calculation. This will be discussed further in the experiments section, as will our application of Heinsen Routing.

5 Experiments

This section contains the following.

Figure 1: Basic model flow



5.1 Data

The three datasets utilized for this project were, as mentioned above, the Stanford Sentiment Treebank (Socher et al., 2013), the Quora Question Pairs dataset (Fernando and Stevenson, 2009), and the SemEval Semantic Textual Similarity dataset (Agirre et al., 2013). For the SST dataset, we were tasked with determining whether or not a sentence fell into one of the five categories: negative, somewhat negative, neutral, somewhat positive, and positive. For the QQP dataset, our task was to determine whether two sentences are paraphrases of each other (this was a binary decision, either they were or they weren't). Finally, for the STS data, we had to determine the strength of semantic similarity between two sentences, ranging from 0 to 5. Note that unlike the previous two classification tasks, this was a regression task.

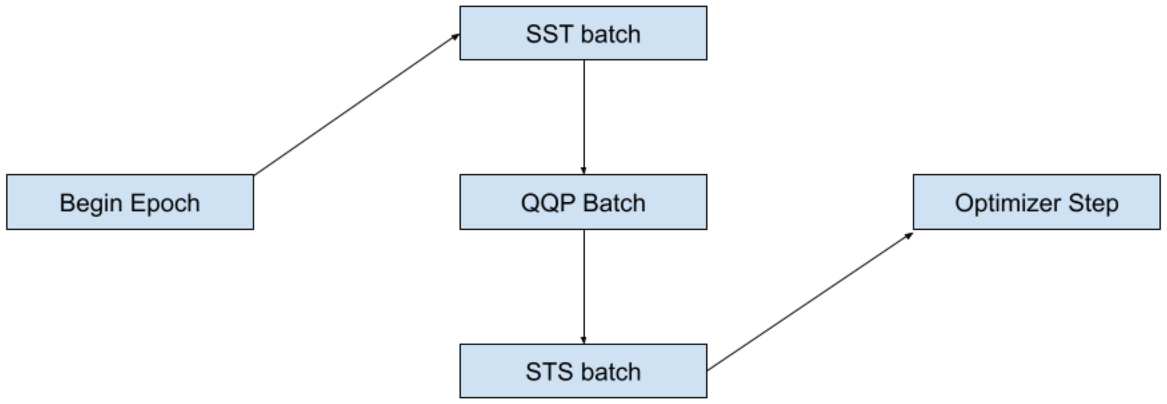
5.2 Evaluation method

The SST dataset was evaluated based on accuracy, whether or not the predicted label matched the true label. The same holds for the QQP dataset, where our label was matched against the binary classification labels provided. Finally, for the STS dataset, we utilized Pearson correlation between the predicted and true similarity scores.

5.3 Experimental details

We ran a total of seven experiments, with their details provided below. The general setup ran finetuning with a learning rate of $1e-05$, 10 epochs, and SMART regularized optimization turned

Figure 2: Training loop



off. For each experiment below, please assume that there were no changes made to this setup unless otherwise mentioned.

5.3.1 $BERT_{BASE}$

We performed finetuning using a very simple extension of the $BERT_{BASE}$, by simply calculating the logit for each of the three tasks directly from the embeddings. This was a very crude baseline, and utilized solely for that purpose.

5.3.2 $BERT_{Multi}$

We then performed the same finetuning, using the $BERT_{Multi}$ model discussed above, and used the results from this experiment as a more rigid baseline for the following experiments.

5.3.3 $BERT_{Multi}$ Parts 2, 3, and 4

While the original $BERT_{Multi}$ only utilizes one intermediate layer, as mentioned before consisting of one Linear, one GELU, and one Dropout layer, we also ran experiments with larger models. Specifically, we ran the same finetuning setup using 2, 3, and 4 intermediate layers, in an attempt to achieve higher performance.

5.3.4 $BERT_{Multi}$ + SMART

Since it has been thoroughly noted that, as models have become more complex, they have needed more and more regularization, we thought it would be useful to implement SMART regularization in an effort to not only improve performance, but also to counteract increases in model complexity. This experiment was only conducted using the baseline $BERT_{Multi}$ model, reasons for which will be discussed further in the analysis section.

5.3.5 $BERT_{Multi}$ + Routing + SMART

This experiment utilized an adaptation of the $BERT_{Multi}$ model, which implemented a Heinsen Routing layer instead of the intermediate layer for the SST task. Similarly to $BERT_{Multi}$ + SMART, this was only attempted once, to be further discussed in the analysis section.

5.4 Results

The following table details our results on the dev set. The model that was submitted to the test leaderboard was an implementation of $BERT_{Multi}$ with two intermediate layers, and will be referred to as $BERT_{Multi_{ekj}}$ to match the submission. $BERT_{BASE'}$ here refers to the barebones version of our multitask $BERT$ that was built on top of the $BERT_{BASE}$ implementation. The subscript to each version of $BERT_{Multi}$ denotes the number of intermediate layers used.

Model	SST Accuracy	QQP Accuracy	STS Correlation
$BERT_{BASE'}$	0.349	0.628	0.526
$BERT_{Multi_1}$	0.495	**0.870**	0.799
$BERT_{Multi_{ekj}}$	**0.512**	0.866	0.856
$BERT_{Multi_3}$	0.503	0.854	**0.857**
$BERT_{Multi_4}$	0.511	0.851	0.851
$BERT_{Multi}$ + SMART	0.504	0.836	0.691
$BERT_{Multi}$ + Routing + SMART	0.262	0.625	0.045

Table 1: Dev Results

Due to its highest average performance across the three separate tasks, $BERT_{Multi_{ekj}}$, or $BERT_{Multi_2}$, was chosen for submission to the test leaderboard, achieving an SST accuracy of 0.487, a QQP accuracy of 0.867, and a STS correlation of 0.848, for an overall test score of 0.759. Due to factors discussed in the following section, these results are better than expected. Initially, we were convinced that implementing SMART regularized optimization would be the most beneficial use of our time and resources, however due to issues during the implementation portion of that phase, and the routing phase, of the experiments, we ended up utilizing just a more complex model for our testing, which performed better than expected.

6 Analysis

We will use this section to discuss the decision to submit a simple, but more complex, version of the multi-headed $BERT$ model over the versions with additions. First and foremost, this was a results-oriented project, tasked with achieving the highest-possible score on the dev and test leaderboards. To that effect, from very early on in the project it was our goal to identify methods that achieved state-of-the-art results on the three benchmark datasets, and replicate those results. In doing so, we hoped to achieve amongst the highest performance on the project's leaderboards. As such, recognizing that SMART regularized optimization was the state-of-the-art or part of the state-of-the-art models for the QQP and STS benchmarks meant that it was meant to be a key part of our implementation. Heinsen Routing was also idenfied in a similar manner, due to our general poor performance on the SST dataset (though we also recognize that state-of-the-art standard on this is much lower than the other two). However, breakdowns in implementation and constraints on compute power meant that decisions had to be made with regards to chasing higher performance.

Jiang et al. (2020)'s reference implementation for their SMART algorithm is quite easily adaptable to single-embedding heads. As a result, it is quite likely that during our attempts to retrofit their implementation to fit our needs for the QQP and STS data, which were pairwise evaluations, we introduced errors into the process that tanked the performance on the dev set. And, as such, due to issues with the extended runtime brought about by the added complexity during finetuning, we decided to pursue alternative routes, after realizing that our attempt had failed. A similar occurrence likely caused the issues with the Routing + SMART model, however it is unclear to us why the STS correlation suddenly became so low. Regardless, had the Routing model achieved higher performance on the SST dataset, we would have been likely to attempt some sort of ensembe model. However, as you can see, there was a decrease in SST accuracy. In fact, we might have been better suited with a random initialization and random classification from the start, judging by our scores on that specific task.

There are some comments we would like to make about our successful models, and some topics that would have required further investigation had we had the time. Firstly, the order in which we trained the model remained the same throught all of our experiments. We wonder if we would see performance spikes or dips depending on the order of the three tasks during training. Secondly, we saw a significant overall performance bump moving from one intermediate layer to two, however this was not true while moving from 2 to 3 or from 3 to 4. We are curious if this pattern continues, or what influence proper regularization would have had. Finally, we utilized cosine similarity and regression loss throughout the whole project, which was indeed a recommended addition, however we wonder what other experiments might have been run on that part of the project.

7 Conclusion

In conclusion, while we did not achieve the peak performance that we were aiming for, we are satisfied with the efforts of our final model with respect to attempting to rank higher on the leaderboards. While failures and time and resource constraints made it difficult to achieve the results we would have wanted, with the methods we would have wanted, we feel as if this project was still successful. In the end, it was our pursuit of higher scores which led us to implement the $BERT_{Multi_2}$ version as our final model, in order to achieve the highest possible ranking that we could, because in the end, performance is all that matters.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Samuel Fernando and Mark Stevenson. 2009. A semantic similarity approach to paraphrase detection. *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*.

Franz A. Heinsen. 2022. An algorithm for routing vectors in sequences.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.