Three Heads are Better than One: Implementing Multiple Models with Task-Specific BERT Heads

Stanford CS224N Default Project

Matt Kaplan
Department of Computer Science
Stanford University
mkaplan2@stanford.edu

Prerit Choudhary
Department of Computer Science
Stanford University
preritc@stanford.edu

Sina Mohammadi
Department of Computer Science
Stanford University
sina24@stanford.edu

Abstract

MinBERT, a variant of the widely-used and widely-applicable BERT model, excels in delivering high performance while utilizing significantly fewer resources, establishing itself as an optimal lightweight language model. Its adaptability renders it suitable for a broad array of applications. This paper delves into three specific use cases. Our project focuses on incorporating the multi-headed self-attention and transformer layers from the original BERT framework, alongside a series of optimization strategies. These include multi-task finetuning, gradient surgery, weight-independent training, sentence-pair concatenation, and cosine similarity enhancements to bolster model efficiency across sentiment analysis, paraphrase detection, and semantic text similarity tasks. The implementation of these techniques varied in their impact on model performance. Notably, training models without sharing weights and leveraging sentence pairs from datasets for embeddings proved most advantageous. Conversely, multi-task finetuning and the application of cosine similarity were found to be less impactful.

1 Key Information to include

- Mentor: Josh Singh
- External Collaborators: N/A
- Sharing project: No

2 Introduction

MinBERT, a variant of the prominent BERT model, has proven itself as a lightweight and efficient large language model, useful for tasks such as text classification, sentiment analysis, text generation, and more. Despite the influx of research and effort being poured into large language models over the past few years, there is still room for improvement. Since the original BERT paper Devlin et al. (2018) was released in 2018, there have been many efforts to improve on this model, with wide-ranging approaches and varying degrees of success. Examples of such improvements include implementations of multi-task learning Bi et al. (2022), as well as attempts to improve regularization through methods such as weight decay regularization Loshchilov and Hutter (2019). From reading a number of these papers, it becomes clear that the process of improving these models is as much an art as it is a science. This creative process motivates much of our work, as we gain inspiration from a mixture of many papers and strive to incorporate many of these techniques into our own model.

The large language model space continues to grow more relevant, and the opportunity to contribute to these cutting edge models lends itself to a challenging yet exciting task, which is the angle form which we approach our own model. Along these lines, our approach significantly revolves around iteration and experimentation, trying a wide range of extensions, including multi-task finetuning, gradient surgery, sentence-pair concatenation, and many others. Throughout the process, we used our experimentation to guide our next steps, ultimately gaining better information about the extensions which improve our model most. By trying a wide variety of extensions and gathering data about which ones complement each other well, we are able to refine our model to include the extensions that we find most useful, leading us to our final results.

3 Related Work

As previously mentioned, there is an abundance of growing work in the field of large language models, including many papers with a focus on BERT. Quite simply, many of these papers expand on or reference the original BERT paper Devlin et al. (2018). Unlike previous works, BERT pre-trains bidirectional representations from unlabeled text, through a joint conditioning on both the left and right context in all layers. As a result, BERT can be fine-tuned using only one further output layer. Ultimately, BERT represented a large breakthrough, being a simple yet powerful model capable of excelling on a wide variety of language-related tasks, including language inference and sentence classification.

In the years since the release of BERT, much work has been done to expand on the original model. While the sheer amount of research makes it infeasible to cover all these papers in detail, here we will highlight a few relevant works for the context of our paper.

One significant extension comes in the form of multitask learning, as detailed in (Bi et al., 2022), which applies both multi-task learning and gradient surgery to the application of news recommendations. While multi-task learning and gradient surgery were both known methods at the time of this 2022 paper, it serves to further highlight the range of applications under which these models can perform. More specifically, the paper further evidences the fact that these methodologies can be leveraged to address the complexities of multi-field information, using relatively novel techniques to enhance domain specific model performance.

Another relevant paper Yu et al. (2020) covers gradient surgery more explicitly, seeking to address the fact that multi-task learning is significantly more difficult than single-task learning. The paper proposes a form of gradient surgery that projects a task's gradient onto the normal plane of the gradient of another task with a conflicting gradient. Ultimately, they saw significant improvements in efficiency and performance, serving as a large motivator for our previous implementation of gradient surgery.

A final paper we will highlight here is seen in (Jiang et al., 2019), which largely focuses on many of the issues seen from traditional approaches to fine-tuning. In particular, aggressive fine-tuning can often cause the model to overfit the training data of downstream tasks, making it difficult for the model to generalize well to unseen data. Techniques discussed in the paper, such as smoothness-inducing adversarial regularization and Bregman proximal point optimization, inspired us to further consider the way which we fine-tune our model, although we ultimately chose not to implement either of these two aforementioned techniques.

As a whole, the work in this space is growing rapidly, making the space an exciting and relevant one for us to explore.

4 Approach

We began by implementing a baseline version of the minBERT model. Specifically, we implemented the multiheaded attention layer and then the other aspects of the transformer layer of the BERT model. The multiheaded attention layer uses a query and a set of key-value pairs to generate an output, where the output is calculated as the weighted sum of the values. Then we completed the transformer implementation by adding an additive and normalization layers with a residual connection and a feed-forward layer. Before we tried any extensions, we tested this baseline model.

The first extension we implemented on top of our baseline was the use of cosine similarity as a loss function for the Paraphrase and STS datasets. This involved first using the CosineEmbeddingLoss() function as a loss function that we applied to each pair of embeddings. We then used cosine similarities between embeddings for each pair as its own layer, normalizing using a linear layer to obtain a logit, and then using a mean squared error (MSE) loss function on that similarity.

Satisfied with the use of cosine similarity as its own layer, we implemented multitask finetuning, first checking its performance without gradient surgery. Because the datasets are different sizes, we determined the number of batches by the size of the smallest dataset. For the larger datasets, we also tried sampling batches randomly rather than always taking the first 6,040 samples, which equates to the size of the smallest dataset. We then combined the losses from each dataset in a few different ways: first, by taking the summation of the three losses; second, by weighing the similar tasks - paraphrasing and semantic similarity - more heavily; and third, by using a loss normalization technique proportional to their magnitudes.

While we wanted to use multitask finetuning to find underlying patterns common across the three datasets, we needed to add gradient surgery to mitigate, though not fully eliminate, the effects of colliding gradients. We tried to do this task by hand before using the $Pytorch/PCGrad$ library.

Next, realizing that our use of cosine similarity was not beneficial, we instead tried to combine our pairs of sentences and separated them with a Separation Token, an idea we got at office hours. We then passed this combined embedding into our layers. We did this for both the Paraphrase and STS datasets, using MSE as the loss functions for both.

At this point, our model improved significantly. Although we knew from the literature that the AdamW optimizer is generally best-suited for models like minBERT, we wanted to quickly check whether using another optimizer, or wrapping the Lookahead Optimizer around AdamW, would improve model performance.

With multitask finetuning only marginally helping but the training on the Sentiment classification dataset helping significantly, we decided to train our model on all three datasets separately. Training on all three datasets consecutively helped, but we realized that sharing the weights might hurt performance, because the model was performing worse on each previous task after being trained for a new one.

Here, instead of somewhat overwriting our weights from the previous tasks when training on a new one, we decided train three separate model heads on the three datasets. We decided to initialize three of the BERT Model objects, one for each task. We then used the separate models as part of the multitask architecture within the forward function by calling the right BERT model's forward function depending on which task called the forward function. We did this by simply adding in a task parameter in forward which was filled by each respective predict function with either 'sentiment', 'paraphrase', or 'similarity'. We then also used separate layers to convert the embeddings to logits for each predict function to keep all models separate. This required using two separate layers for the concatenated embedding datasets and using unshared weights between the three tasks.

To limit our training times, we first only ran one epoch. Then, realizing that training the Paraphrase dataset took substantially longer than training the other two, we decided we would run five epochs for the sentiment classification and STS datasets, and only one on the Paraphrase dataset. This allowed us to try more techniques while still improving our model's performance. We wanted to further improve our accuracy and also were hoping to reduce our runtime. As a result, we decided to double our learning rate from 1E-5 to 2E-5 and increase our batch size from 8 to 24. Increasing the batch size helped reduce training time as it made more use of the parallel computing power within the V100 colab GPU.

5 Experiments

5.1 Data

We use multiple datasets throughout our project. For sentiment analysis, we use the Stanford Sentiment Treebank [1] (SST), which includes 11,855 single sentences extracted from movie reviews,

[1] https://nlp.stanford.edu/sentiment/treebank.html

with each phrase receiving labels ranging from negative to positive. We also use the CFIMDB dataset, which contains 2,434 highly polarizing movie reviews, with each movie containing either a negative or positive binary label. For both these datasets, we utilize pre-trained weights and fine-tuning to optimize performance.

For extending and improving downstream tasks, we also use three datasets. For the sentiment analysis task, we again use SST. For paraphrase detection, we use the Quora dataset [2], which consists of 400,000 question pairs, with labels denoting whether the questions in the pair are paraphrases of one another. Finally, we use the SemEval STS Benchmark dataset (Agirre et al. (2013)) for semantic textual similarity; this dataset includes 8,268 sentence pairs that range from 0 (unrelated) to 5 (equivalent meaning) on a scale of similarity to one another.

5.2 Evaluation method

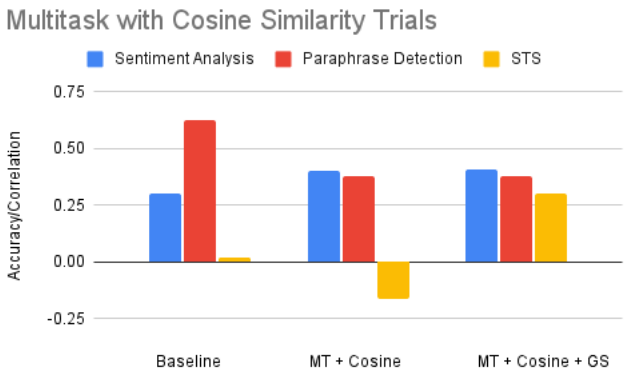
As an evaluation metric, we primarily rely on the default metrics presented by the project handout. Thus, we mainly use accuracy. Note that when testing the SemEval STS Benchmark Dataset, we use the Pearson correlation of the true similarity values against the predicted similarity values, as described in the original SemEval paper (Agirre et al. (2013)).

5.3 Experimental details

We ran our experiments on Google Colab, primarily using a V100 GPU. We initially attempted to run our code locally with a GPU, but after encountering issues, we pivoted to using CUDA for GPU performance acceleration. Ultimately, we saw best results with training time by using Google Colab with a V100 GPU. We had a training time of roughly 3.5 minutes per epoch for the STS dataset. While we ran our experiments several times in order to test various extensions and in an attempt to gather data about optimized hyperparameters, some details about our model configuration and experimentation are as follows. Our model had a hidden size of 768. For training, we used a dropout rate of 0.0 (after experimenting with dropout rates ranging from 0.0 to 0.3). We completed most runs of our model with a learning rate of $1e-5$ and a bath size of 8. However, after further research, we found strong evidence to suggest that increasing both the learning rate and batch size could be helpful, utilizing ideas discussed in Smith et al. (2017). Thus, for our final run, we increased the the learning rate to $2e-5$, also scaling up the batch size to 24. The training time for our model varied depending on implemented extensions and various hyperparameters; however, on average, the training time was roughly 2 hours.

5.4 Results

We ran several trials with a variety of implemented extensions and hyperparameters. Below are much of our results visualized. Note that we use "MT" to denote multitask fine-tuning, "GS" to denote gradient surgery, "CT" to denote consecutive training, "ST" to denote our implementation with separation tokens, "PD" to denote the Paraphrase data, and "LR" to denote the learning rate:



[2]https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

Figure 1

The above graph shows the progression of our results from baseline to our implementation with multitask finetuning, cosine similarity, and gradient surgery. We obtain fairly expected results of 0.303 accuracy for sentiment analysis, and 0.02 correlation for the semantic textual similarity (STS). Notably, our accuracy of 0.624 for the Paraphrase detection exceeded our initial expectations. However, we can see that the accuracy for Paraphrase detection declines as we added multitask finetuning and gradient surgery, which presented a slightly unexpected shortcoming for us, likely due to the fact that we had to remove much of our data from the large Quora dataset to even out the number of batches for each task; however, this also causes us to lose valuable data. With multitask finetuning and cosine similarity and gradient surgery, we saw marginal improvements on the other tasks, obtaining 0.404 accuracy for sentiment analysis and 0.300 correlation for STS. However, these improvements still left us short of our desired goal, leading us to continue implementing other extensions. This continual process of searching and improving highlights our emphasis on art throughout our approach.

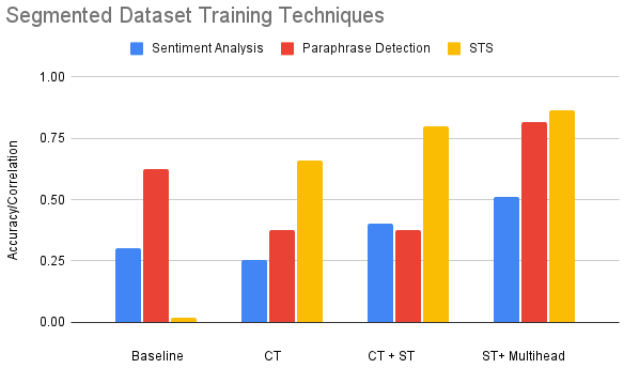


Figure 2

The above figure shows our model performance with various forms of consecutive training, separation tokens, and multihead finetuning. Notably, we obtain our best model to-date through separation tokens and multihead finetuning, reaching accuracies of 0.512 and 0.816 for the sentiment analysis and Paraphrase detection, respectively, along with a correlation of 0.866 for the STS. This improvement aligns with our expectations, as we anticipated a significant performance boost through both multitask finetuning and the use of separation tokens.

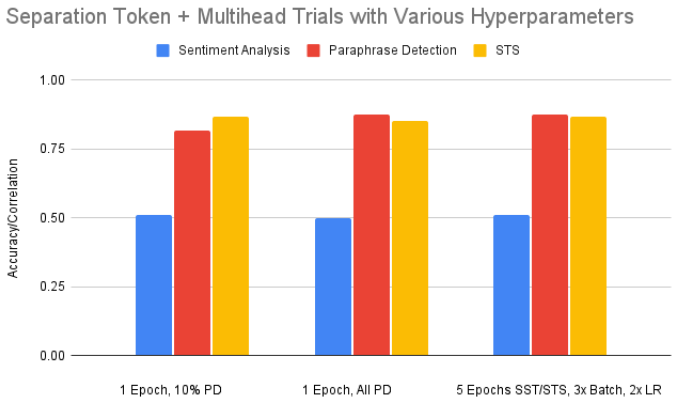


Figure 3

Seen above, Figure 3 primarily shows the progression of our model results with various hyperparameters and factors. We see tradeoffs between performance time and accuracy through the percentage of the Paraphrase dataset that we use, along with the number of epochs. As a whole, we obtain similar results with these various tweaks, with our sentiment analysis accuracy going from 0.512 to 0.500 to

0.509. Similarly, the accuracy for the Paraphrase detection goes from 0.816 to 0.874 to 0.875, and the STS correlation goes from 0.866 to 0.853 to 0.868.

Ultimately, we obtained accuracies/correlations for the test leaderboard as follows:
SST Test Accuracy: 0.531
Paraphrase test accuracy: 0.873
STS Test Correlation: 0.854
Overall test score: 0.777

6 Analysis

Our results can be evaluated in three phases. Phase One was the use of two different multitasking techniques with cosine similarity; Phase Two was the use of a separation token and multiheading techniques; and the third was refining the proven techniques from phase two and testing hyperparameter combinations.

In the first phase, we found that the multitask finetuning approaches with cosine similarity did not help our baseline. One challenge with using multitask finetuning and combining the losses is that our implementation required that we use the same amount of data from each dataset. The datasets are vastly different sizes, with the paraphrase dataset being by far the largest. Using only a fraction of that data in our multitask finetuning likely hurt performance, which is why we saw comparatively worse performance scores for the Paraphrase dataset relative to the others on multitask finetuning. Gradient surgery was able to aid our results slightly by mitigating the effects of clashing gradients.

But gradient surgery cannot cancel out the effects of overriding weights, which is why we then decided to train the model on different tasks separately. By not sharing weights, we have individual model heads that are specifically trained for those individual tasks. While this is more computationally expensive, it will lead to better results. While we might expect one model - after multitask finetuning and training on all three sets - to perform better because it can rely on underlying patterns that are consistent across all three tasks, unexpectedly, the multiheaded approach of not sharing weights proved more effective. Another possible explanation for this is that the tasks, while similar, are distinct enough that multitask finetuning cannot identify significant patterns between the tasks that make the weights worth sharing. Further, multitask finetuning relied on our combining the losses from the three tasks. While we tried a few different approaches in combining the losses to maximize our overall score, we likely did not find one that accurately weighed improvements in the three tasks equally such that the overall score could be maximized. Our approach could have given more time to this task, but we concluded it would only benefit our results marginally, which is why we tried a new approach altogether.

In the second phase, we focused on adding two techniques: the first was concatenating the embeddings with a separation token, and the second was using the multiheaded approach with no weight sharing. Using concatenation with a separation token for both the Paraphrase and STS datasets produced results that reflected a surprising improvement. One possible explanation is that, for STS, concatenating embeddings with a SEP token provides a single, continuous input, which may allow for better semantic similarity comparisons than looking at them as purely separate inputs. Further, the cosine similarity formula in $torch.nn.CosineEmbeddingLoss$ simply looks at the angle between the two embeddings, which could lose more fine-grained relationships that are captured when the sentences are fed as a single input to the model. The second part of phase two, changing the model to have separate BERT heads for each task, was likely an effective approach because we had seen the effects of training on all three datasets with weight sharing and without gradient surgery. When datasets were trained consecutively, we found that the dataset trained last of the three performed the best comparatively. This occurrence can likely be attributed to the fact that the weights' values were overriden every time we train on the next dataset. Although there are resource costs associated with training the weights fully separately, we get significantly better performance results.

In the third phase, after finding a combination - concatenating embeddings for Paraphrase and STS datasets and using a multiheaded approach without weight sharing - that worked well, we focused on merely tweaking our existing approach for more modest improvements. Here, we saw improvements on Paraphrase performance when we trained on all of the Paraphrase data, rather than only 10 percent

of it (which we had done to save training time). This improvement makes sense given that more training data on a task usually leads to better outcomes on that task. We then saw further improvement when we increased the number of epochs on the Sentiment and STS dataset trainings, because we were giving the model more chances to learn from the data and adjust its weights. We also tripled the batch size to cut down our training time, because the GPU handles an entire batch at once and allows for parallelization, so a larger batch size will speed up the training process at the cost of GPU RAM. One reason that increasing the batch size likely did not hurt performance was that we also doubled the learning rate (increasing the step size), allowing the model to adjust its weights more aggressively, which is suited for a larger batch size that can give more confidence in the adjustment the model is making.[3]

7 Conclusion

This project implemented various techniques and tweaks to improve on a baseline minBERT model. It implemented these techniques and tweaks to improve minBERT's performance on three tasks: Sentiment Analysis, Paraphrase Detection, and Semantic Text Similarity. These techniques were often informed by the tasks themselves, and some targeted specific tasks while not intending to have an effect on others.

There are several main findings from our three phase evaluation. First, multitask finetuning with cosine similarity did not significantly improve performance, though adding gradient surgery helped multitask finetuning perform slightly better. Second, introducing the concatenated embeddings with a separation token led to surprising improvements, particularly with the STS dataset, suggesting a more nuanced capture of semantic relationships compared to using embeddings as separate inputs. Implementing separate BERT heads for each task (multiheaded approach without weight sharing) also significantly improved results, as it avoided the overriding of weights that was detrimental in multitask finetuning. Third, using all available data, increasing the batch size, and doubling the learning rate all led to modest improvements built upon our high performance model.

Our two main achievements were the use of the concatenated sentences with the separation token and the use of a multiheaded approach with no weight sharing. The combination of using these two techniques performed far better than multitask finetuning, suggesting an important finding: the way we structure our data can sometimes improve performance more than attempts at finding underlying patterns consistent across multiple datasets.

There are several limitations to our work worth noting. First, our iterative approach to developing a high performance model did not test all possible permutations of our different techniques, limiting our confidence which techniques helped and which did not. For example, we never attempted to use multitask finetuning without cosine similarity, meaning that we might have found multitask finetuning more effective without a cosine similarity layer. In another example, we increased the batch sizes and the learning rate together, confounding the two variables and limiting our ability to say which led to greater performance. Our approach mostly reflected our desire to implement several improvements at once to chase high performance, but at times that sacrificed knowing which techniques or tweaks were truly most effective. Another limitation of our work is that, to reduce training times, we trained for a reduced number of epochs and an unequal number of epochs on each task. We did this to cut down training times and reduce our resource footprint, but we likely lost quality in performance as a result. This also reduces the efficacy of comparing our approach with those of researchers who trained for more epochs.

There are several avenues for future work that can be built upon our project. First, one could further explore whether multitask finetuning might work with some of our later approaches. For example, we could apply multitask finetuning with the use of concatenated sentences, which may perform better than multitask finetuning with cosine similarity. This could also involve investigating how similar or different the tasks truly are to better understand when multitask learning is beneficial, or it could involve developing more sophisticated methods to combine losses from different tasks. Second, following our multiheaded model with no weight sharing approach, we could conduct a more exhaustive hyperparameter search to find a more exact optimal performance. Third, considering that we reduced training times on our approaches that had the highest performance, we might consider

[3]Smith et al. "Don't decay the learning rate, increase the batch size."

further analysis that explores the trade-offs between resource costs and performance gains in the multiheaded approach.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, , and Weiwei Guo. 2013. *sem 2013 shared task: Semantic textual similarity. pages 32–43, Online. Association for Computational Linguistics.

Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. 2022. Mtrec: Multi-task learning over bert for news recommendation. page 2663–2669, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. pages 01–16, Online.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *The 58th annual meeting of the Association for Computational Linguistics*, pages 01–20, Online.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. pages 01–19, Online. Association for Computational Linguistics.

Samuel Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. 2017. Don't decay the learning rate, increase the batch size. pages 01–11, Online. ICLR.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. pages 01–27, Online.