Effects of Appropriate Modeling of Tasks and Hyperparameters on Downstream Tasks

Stanford CS224N Default Project

Adrian Gamarra Lafuente
Department of Computer Science
Stanford University
a1090588@stanford.edu

Avi Udash
Department of Computer Science
Stanford University
udashavi@stanford.edu

Abstract

BERT (Bidirectional Encoder Representations from Transformers) (Delvin et al. (2019)) is a pre-trained language representation model that can be fine-tuned to achieve state-of-the-art results on downstream tasks. We explore possible extensions on our minBERT model on three downstream tasks of sentiment analysis, paraphrase detection, and semantic textual similarity. We will explore further pre-finetuning, effects of freezing top layers, different learning rates, varying dropout probabilities, and task-specific architectures to test whether they can help improve our minBERT model on the three downstream tasks. We will find that freezing layers can lead to decreased training times while maintaining similar performance, that having a model that appropriately models that task at hand can have dramatic increases in performance, and that altering hyperparameters offers negligible to little improvement in performance.

1 Key Information to include

- Mentor: Annabelle Wang
- Team Contribtions:
  - Adrian Gamarra Lafuente: Writing of paper, implementing components of minBERT, exploring normaliztion layers, freezing of layers, extra added layers, ethics, research.
  - Avi Udash: Writing of paper, cloud computing, implementing components of minBERT, exploring learning rates, dropout probabilities, implementing tensorboards, tracking expirement results, and uploading model for submission.

2 Introduction

Delvin et al. (2019) introduced BERT as a pre-trained language representation model that can achieve state-of-the-art results by fine-tuning with single output layer on top of BERT. They demonstrated that BERT was able to to achieve a GLUE score of 80.5%, a MultiNLI accuracy of 86.7%, a SQuAD v1.1 question answering Test F1 of 93.2, and a SQuAD v2.0 Test F1 of 83.1. BERT pushed the boundary for language models thanks to its bi-directional pre-training with a masked language model pre-training objective.

We leverage the power of BERT by exploring a multi-task approach to simultaneously perform three downstream tasks of sentiment analysis, paraphrase detection, and semantic textual similarity from the same model. Multi-task models allow for a more efficient use of data as data for one task, can simultaneously train the base model for the other tasks. This generalized approach can leverage knowledge from all the tasks as to improve overall performance. As well, it allows for a single model to perform different tasks, which reduces complexity as less models are needed.

Not only will we implement our multi-task model, we will also explore different improvements that can be made as to maximize its overall score. We will explore further fine-training as to better teach the model for our downstream tasks. Furthermore, we will explore the effects of freezing the top layers of the BERT model during fine-tuning to reduce training time. Then we will test out different hyperparameters, like learning rate and dropout rate, as well as model architectures with added layers and task-specific additions.

Additionally, we decided to focus on working with a single 12 layered BERT model. We're aware that using a larger model can lead to better results, but we wanted to explore the potential of working with a single model as it is less computationally expensive and requires less overhead.

Ultimately, we would find that appropriately modeling the specific tasks had a dramatic increase in our overall score. Unfortunately, we would show that although altering hyperparameters could lead to an increased performance in our dev set, the model failed to show improvements in our test set. This is a symptom of overfitting, as BERT has million of parameters and we were working with less data for a number of our downstream tasks.

3 Related Work

Sun et al. (2019) also looks into how the BERT model can be leveraged with fine-tuning to perform downstream tasks. They used the official BERT model, which consists of an embedding layer, followed by 12 transformer layers, and a pooling layer. We will use the same architecture for our base BERT model. The authors would find that further fine-tuning on in-domain and within-task data can significantly boost performance on downstream tasks. Moreover, the authors explore the effects of freezing layers and found that freezing the top layer is the most useful for text classification. We will also explore how freezing multiple top layers can have varying effects and be useful to decrease training times. Lastly, the authors found that a low learning rate, such as 2e-5, is needed to overcome catastrophic forgetting.

Zhang et al. (2017) takes a look at deep learning models as a whole to interrogate how they are sufficient to memorize training data. They find that explicit and implicit regularizers do help, but aren't the primary reason why a model may be over-fitting. We will use these findings as we will see that our fine-tuned models run into over-fitting early on because of our limited data. With this knowledge, we will take a look at our model architecture, instead of relying solely on regularization methods for improve performance.

4 Approach

Our approach is broken up into two components, which will be tested on three downstream tasks of sentiment classification, paraphrase detection, and semantic textual similarity. The first component is the implementation of minBERT and fine-tuning for sentiment analysis, a single task. The second component is implementing a multi-task classifier for two additional tasks and exploring different potential improvements.

First, we implemented minBERT with the help of boilerplate code provided to us by Stanford University. We implemented the multi-head self attention, the BERT transformer layer, the classifier for sentiment analysis, and an Adam optimizer. We would then both test the results from training just the added classifier and training whole minBERT model and the classifier on the SST dataset. We were able to conclude that our minBERT model was working appropriately after running these tests.

Then, we implemented a multi-task classifier to support the two additional tasks on top of sentimental analysis and train from multiple datasets. We achieved this with the help of some boilerplate from Stanford University. After implementing this, we are able to further fine-tune our model on the Quora and SemEval STS Benchmark datasets as these are curated for the paraphrase detection and semantic textual similarity.

When conducting experiments, our base multi-task model architecture consisted of a BERT model with 12 layers and a hidden size of 768. Additionally, we added a common dropout layer and a common linear layer, so our model will be able to learn patterns between the three tasks. We will explore this decision later on. Then for each task, we had an additional task-specific head with the correct number of output nodes. While training, our hyper-parameters were, unless otherwise

specified, a learning rate of $2e-5$, an exponential decay of 0.9, a dropout probability of 0.3, a batch size of 32. We also decided to keep the BERT model frozen, apart from the top 2 BERT layers and the final pooling layer for faster computation. We will also explore this decision later on. We ran each experiment with 10 epochs, but only saved the model weights that resulted in the best overall dev score. We conducted our training on an Nvidia Tesla V100 GPU, with each epoch taking around 7 minutes with this configuration. A figure of our model architecture is presented below.

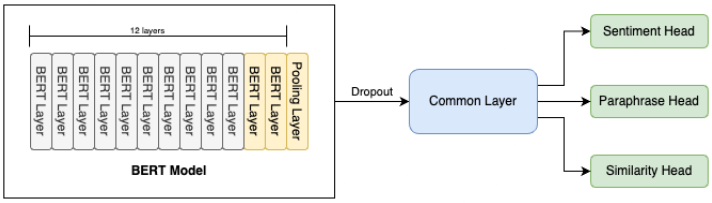

Figure 1: Our expiremental multi-task model architecture. The yellow layers represent the BERT layers that were unfrozen during our training.

Each task had its own separate head that handled the input and output appropriately. The sentiment head consisted of a single extra linear layer to match the number of labels. The paraphrase head consisted of running the two inputs separately on the same BERT model, then concatenating the result and passing it through a final linear layer for classification.

The similarity head went through two iterations. At first, we had done the same approach for predicting paraphrase. This would prove to be detrimental to overall performance. Initially, we believed that it was an over-fitting issue, but as seen by Sun et al. (2019), this wasn't the whole problem. Therefore, we decided to take a step back and re-approach our tasks at hand. We felt that for semantic textual similarity, it was unreasonable to run the inputs through our BERT model separately since BERT captures contextualized meanings between words, meaning it can struggle to classify on a linear scale. Therefore, if we wanted to classify two sentences varying similarity to each other, we should concatenate them before entering them into the BERT model. This approach proved highly successful and increased our performance by roughly 10 percent. We would end up using this second iteration for the rest of the experiment. We would carry this finding as an important avenue for performance increase.

5 Experiments

5.1 Data

We will be using 3 different datasets while we train our model on the three downstream tasks. The 3 datasets are as follows:

1. The Stanford Sentiment Treebank (SST) (Socher et al. (2013)) includes sentences where each phrase has a label of negative, somewhat negative, neutral, somewhat positive, or positive. They are broken up as 8,544 examples for training, 1,101 sentence for dev, and 2,210 for test. This dataset will be used for the sentiment classification task.

2. The Quora dataset (Iyer et al. (2017)) includes question pairs with binary label of whether one is the paraphrases of one another. Our data contains 141,506 examples pairs for training, 20,215 pairs for dev, and 40,431 pairs for testing. This dataset will be used for the paraphrase detection task.

3. The SemEval STS Benchmark dataset (Agirre et al. (2013)) includes 8,628 sentence pairs of varying similarity on a scale 0-5. It's broken up to 6,041 examples pairs for training, 864 pairs for dev, and 1,726 pairs for test. This dataset will be used for semantic textual similarity correlation.

5.2 Evaluation method

In order to evaluate the method, we ran predictions for each task on the respective dev and test sets. We use accuracy for the SST and the Quora datasets (sentiment classification and paraphrase detection tasks) and we use Pearson correlation for the STS dataset (semantic similarity task). Once we get the three different scores, we can get an overall score by normalizing the Pearson correlation for STS to range from 0 to 1 (Pearson correlation outputs range -1 to 1 by default), and then averaging the three normalized numbers with equal weight to get the overall score.

For training, we use a different loss for each task. For the SST dataset (predict sentiment task), we used cross entropy loss. For the Quora dataset (predict paraphrase task), we use binary cross entropy with logits loss. For the STS dataset (predict semantic task), we were considering using cosine similarity loss, but decided to use mean squared error as it was used in Reimers and Gurevych (2019).

5.3 Experimental details

We explored a variety of options for how to improve our model and ran experiments to test for performance. We hoped to be able to make improvements to our model efficiently by reducing training time while running experiments to estimate our model's final performance and optimizing the hyper parameters. The detailed descriptions for each experiment is provided in this section.

5.3.1 Experiment 1: Varying number of BERT layers to unfreeze

Rather than unfreezing all of BERT during training, it's possible to only train a certain number of BERT layers, while freezing the rest of the model. Each layer of BERT captures different features of the input text and so we hoped that only training a certain number of encoding layers would drastically reduce training times while still providing close to the same accuracy or possibly even better accuracy. Sun et al. (2019) conducted a similar experiment but did not mention the improved training time, which we feel is a significant reason for this experiment. This would, in the future, allow us to considerably speed up training times for our other experiments. We ran this experiment in three different configurations, with our baseline for the experiment being a model that was completely unfrozen. We chose to experiment with unfreezing up to one, two, and three top layers. In each case, the the pooling layer was always unfrozen.

Our results for this experiment are shown in Table 1.

5.3.2 Experiment 2: Varying learning rate

Another experiment we conducted was varying the learning rate. Catastrophic forgetting, introduced by McCloskey and Cohen (1989), is a common problem where a neural network could essentially "forget" earlier training data while learning new knowledge. Sun et al. (2019) suggested a learning rate of 2e-5 because a significantly higher learning rate would have caused catastrophic forgetting and showed results of high learning rates not converging. However, the paper did not discuss the results of learning rates similar to 2e-5, so we wanted to test how other learning rates, such as 1e-5 and 3e-5, would impact the model while keeping the rest of the hyperparameters and architecture the same. Figure 2 in the results section shows how the learning rates decayed over time and how the model's overall score was impacted.

5.3.3 Experiment 3: Varying dropout rate

Srivastava et al. (2014) introduced dropout as a simple technique to prevent the problem of overfitting in large neural networks, which can happen as a result of low training data on a model with a large number of parameters. We suspected that overfitting was happening on our model because for some experiments, the dev accuracy peaked around epoch 4 or 5 and would decline after that, while the training loss would continue to improve. Larger dropout probability should allow our model to generalize better while training as a regularization strategy, so we wanted to test the impact that dropout had on our accuracy. We ran the following tests, making sure to only modify the dropout probability while keeping the rest of the architecture and hyperparameters the same. Our baseline was set at 0.3 since that was the default number provided to us and we had been training our other experiments with 0.3 dropout probability.

1. Dropout probability of 0.0
2. Dropout probability of 0.3 (baseline)
3. Dropout probability of 0.4
4. Dropout probability of 0.5
5. Dropout probability of 0.6

Our results for this experiment are shown in Table 2.

5.3.4 Experiment 4: Varying number of extra layers to add

For this experiment, we wanted to test how adding simple linear layers would impact the model. We tested a number of configurations for our model. We experimented with adding more common linear layers, as well as more task-specific layers.

The configurations we use for this experiment:

1. Baseline: One common layer + one task-specific output layer per task (112.3M parameters)
2. No common layer + one task-specific output layer per task (111.7M)
3. No common layer + three task-specific layers per task (113.5M)
4. No common layer + five task-specific layers per task (117M)
5. Two common layers + two task layer per task (114.7M)
6. Five common layers + five task layers per task (121.7M)
7. Nine common layers + nine task layers per task (131.2M)

The results are covered in Table 3.

5.3.5 Final Model

Using the results from the above experiments, we will create a final model that aims to utilize the best hyperparameters from our experiments. The model's configuration will be as follows:

- Learning rate: 1e-5 with an exponential decay of 0.9 per epoch
- Batch size: 32
- Dropout probability: 50%
- No common layers + 3 task-specific layers per task

5.4 Results

5.4.1 Experiment 1

The following table shows the results for experiment 1. Unsurprisingly, the fully fine-tuned model outperforms the partially frozen models, but the accuracies are still high all around. However, training times dropped significantly for the models with frozen layers. Using this data, we were able to run our other experiments much quicker while maintaining a similar level of performance. Ultimately, we chose to run our other experiments with only the 2 top BERT layers unfrozen, which had maintained 97% of the performance while reducing training time by 52%!

	Best Overall Score	**Training Time**
Baseline	**0.7365**	2.306 hr
1 Top Layer	0.712	**1.018 hr**
2 Top Layers	0.7147	1.117 hr
3 Top Layers	0.7221	1.218 hr

Table 1: The results from experiment 1. This experiment contained a baseline, unfreezing the top encoder layer, unfreezing the top two encoding layers, and unfreezing the top three encoding layers.

5.4.2 Experiment 2

The graph below shows the data for experiment 2, with varying learning rates. We had expected the learning rate data to be in line with the findings from Sun et al. (2019), which recommended a learning rate of 2e-5. However, through this experiment, we found that 1e-5 was actually the best learning rate in our case. It is possible though, that because we tested this with some BERT layers still frozen, this finding might not correspond to the best learning rate for the full fine-tuned model.

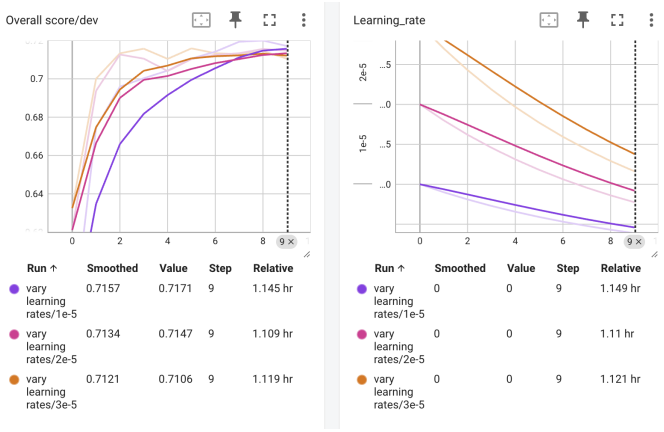

Figure 2: Graphs of data from varying learning rate examples. The graph of overall scores is shown on the left, and the graph of the learning rates are shown on the right.

5.4.3 Experiment 3

A major problem we ran into with our model was overfitting, particularly on the sentiment classification task. The accuracy would peak around epoch 3 or 4, and would subsequently begin to drop in future epochs as the training loss decreased. Because dropout is a method to increase regularization, we had hoped to see that increasing dropout probability would lead to a better model.

Dropout Probability	Best Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Semantic Correlation
0%	0.713	0.471	0.765	0.807
30% (baseline)	0.7134	0.4723	**0.7683**	**0.8072**
40%	0.7157	0.4869	0.7658	0.7992
50%	**0.7159**	**0.4873**	0.7624	0.801
60%	0.7151	0.4859	0.7606	0.8035

Table 2: Results of Experiment 3. This experiment tested the impacts of varying dropout probabilities on our multi-task model. We tested probabilities of 0%, 30%, 40%, 50%, and 60%.

After this experiment, we saw that ultimately, a higher dropout did increase the model's overall score, but only by a small margin. As expected, the biggest gain was realized in the sentiment classification task.

5.4.4 Experiment 4

Experiment 4 was our largest experiment, and it also had the biggest impact. Rather than trying different hyperparametes, we chose to modify our model's architecture by trying out various combinations of the number of common layers and task-specific layers. We found out that the optimal configuration was having no additional common layers and having 3 task-specific layers (including the output layer). Increasing the task-specific layers to five yielded worse results than three layers, which was an interesting finding. Something that we didn't expect from these results was that introducing common layers on top of BERT was actually regressive to the functionality of our model. The worst configuration was 8 common layers with 8 task-specific layers per task.

Configuration	Best Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Semantic Correlation
1	0.7156	0.4768	0.7773	0.798
2	0.7256	**0.5041**	0.7707	0.804
3	**0.7286**	0.4995	**0.7818**	**0.8088**
4	0.7175	0.4768	0.7738	0.804
5	0.7188	0.4841	0.7728	0.7991
6	0.7148	0.4825	0.7651	0.7901
7	0.6936	0.4641	0.7305	0.7722

Table 3: Results of Experiment 4. This experiment tested the varying configurations for our multi-task model as mentioned in 5.3.4.

5.4.5 Final Test Results

The results we got on the test set are as follows:

Model	Test Overall Score	Dev Overall Score	Test Sentiment Accuracy	Test Paraphrase Accuracy	Test Semantic Correlation
Baseline: Pretrain	0.591	0.5834	0.380	0.672	0.439
Further Finetuning	**0.743**	0.736	0.523	0.784	**0.844**
Final Model	0.741	**0.743**	0.510	**0.795**	0.837

We had an huge improvement to our baseline model by further fine-tuning on the three datasets that we received. However, what was surprising was the test results between our other two models. We had expected the final model to do better than the fine-tuned model, and that expectation held for the dev set, since our final model did quite a bit better than the solely fine-tuned model. As well, it was supported by our previous experiments that our additions would lead to increased performance. However, on the test set, our solely fine-tuned model did much better than expected while the final model did slightly worse than expected. Therefore, the solely fine-tuned model actually beat out the final model, which was surprising.

Our approach for this project was to try to figure out an efficient way to fine-tune BERT for multi-task learning. Through the use of choosing n top layers to unfreeze while training and running various experiments to figure out the most optimal settings for fine-tuning BERT, we had hoped to be able to achieve good results.

6 Analysis

Though some of the findings in our results were surprising, we were able to explain why some of the experiment resulted in these numbers. A large problem for us was over-fitting because of the limited data for some of the datasets, particularly in the case of the sentiment classification task. The minimal improvements were in line with the findings from Zhang et al. (2017) that showed that explicit regularization can help, but isn't the main source of improvement for generalization. Instead, we had shown that appropriately modeling the downstream tasks had lead to the largest increase in performance as discussed in the Approach section. Additionally, both models had no common layer, which most likely improved both of their performances. This can be explained as the downstream tasks weren't exactly similar, such as sentiment and semantic similarity. Therefore sharing a common layer would lead to different objectives.

We did show, however, that we could greatly increase training speed while maintaining similar performance levels by simply freezing most of the BERT model and only training a certain number of top layers as supported by Sun et al. (2019). We were able to run more experiments efficiently and were able to save computation and make training more accessible. One thing to keep in mind however, is that the experiments that we ran with frozen BERT layers might not directly correspond to the results that we would have gotten through fully fine-tuning our model. Our learning rate results did not agree with the results that Sun et al. (2019) showed, and a reason for that might have been due to training with a partially frozen BERT. Regardless, our method was a good way to estimate performance during experimentation.

7 Conclusion

In conclusion, our findings showed that to achieve utmost performance on downstream tasks, researchers need to appropriately model the task beforehand and decrease the amount of information shared between the tasks. Moreover, we can see how further fine-tuning of the model leads to increased performance.

Additionally, we demonstrated how freezing the bottom layers of BERT can lead to greatly decreased training times and therefore save computation and make language modeling more accessible. Likewise, it demonstrates the possibility to run more experiments on the same amount of computation.

Some of our limitations derived from working around the BERT model with our limited data. Our data was quickly over-fitted as the size of the model could memorize the training data. Although we showed that it is important to model the tasks at hand appropriately, working with more data and separate models can lead to improved results. This could serve as an avenue for further research.

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

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First Quora Dataset Release: Question Pairs.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

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 Manning, Andrew Ng, and Christopher Potts. 2013. Sentiment Analysis.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

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

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017. Understanding deep learning requires rethinking generalization.