Progressive Layer Sharing on BERT

Stanford CS224N Default Project

Nathaniel Grudzinski
Department of Computer Science
Stanford University
ngrudzi@stanford.edu

Abstract

In this project, I aim to enhance the performance of a BERT multitask classifier by refining the vanilla fine-tuning process. The classifier is designed to handle three distinct tasks: Sentiment Analysis, Paraphrase Detection, and Similarity Correlation. My ultimate objective is to explore various modifications beyond the vanilla approach and identify the most effective changes. The key focus lies in progressively sharing layers between tasks. Specifically, I envision that insights gained from sentiment analysis would be of great use to the other two tasks. Consequently, I feed layers from the sentiment analysis task into the other two tasks, but not vice versa. I also found that sharing the paraphrase task output with the similarity correlation task improved that task further. This approach has yielded modest improvements over the standard fine-tuning method, particularly in the context of the similarity correlation task. That task saw an improvement of 20% in its dev accuracy (50.8% to 60.8% accuracy)

1 Key Information to include

- Mentor: n/a
- External Collaborators (if you have any): n/a
- Sharing project: n/a

2 Introduction

The impetus for this project comes from my idea for a custom project that I eventually had to forgo for the default project because of a number of blocking issues I encountered. The idea for that project was to investigate Progressive Neural Networks (PNNs) and their applications in Parts of Speech tagging. I often try to think about Machine Learning topics and tie them back into how we as humans learn. One thing that has struck me throughout this course and that I wanted to look at with my custom project is this idea of whether gradually increasing complexity in training would improve a model's training. The original idea was to first train on sentences with simple parts of speech, and then gradually add more complex parts of speech as training progressed. I also wanted to look at splitting up a domain into areas of complexity and having each area of complexity be a lane in a PNN with the simple layers feeding into their corresponding more complex layers.

This research interest of mine carried over into this project. I wanted to look into whether sharing the outputs of more foundational tasks would improve the training of subsequent tasks. The tasks in the default project fit this notion very well. Sentiment analysis will certainly be helpful context to determine whether or not two sentences are paraphrases or how similar they are. Also, whether or not two sentences are paraphrases will be helpful in determining how similar two sentences are. This was not an extension listed in the project description, it was developed by me from this broader concept I was developing for the custom project.

This problem is interesting because it is fairly novel. I have not been able to find a research paper that exactly tackles this model architecture. I also was not sure whether or not the outputs of the previous layers would be useful concatenated to the inputs of the subsequent tasks. I didn't know whether or not those subsequent tasks would be able to extract useful features from those outputs. And the results make for a marked improvement, especially for the Similarity task which takes in the outputs of both the other tasks as context. There was a considerable amount of experimenting when it came to my approach. I had a broad idea of how I wanted this to work when starting the project, but wasn't sure of the specifics of what would actually work. I will go more in depth into the types of experiments I ran and what ended up working later in this paper.

3 Related Work

This project is architecturally a multitask classifier built on top of a pretrained BERT model (Devlin et al., 2019). BERT stands for Bidirectional Encoder Representations from Transformers. Unlike previous pretrained language models BERT learns its contextual information from both directions in a sentence. This results in it being adept at capturing more context-dependent nuances in its embeddings.

Bi et al. (2022) is a paper that tackles multi-task learning on BERT in the context of News recommendations. This research aims to finetune BERT on two tasks, category classification and named entity recognition. This paper takes the output along with the title of the news article and creates a new news representation via attentive multi-field learning. This is then fed into a news recommendation framework they define in the paper.

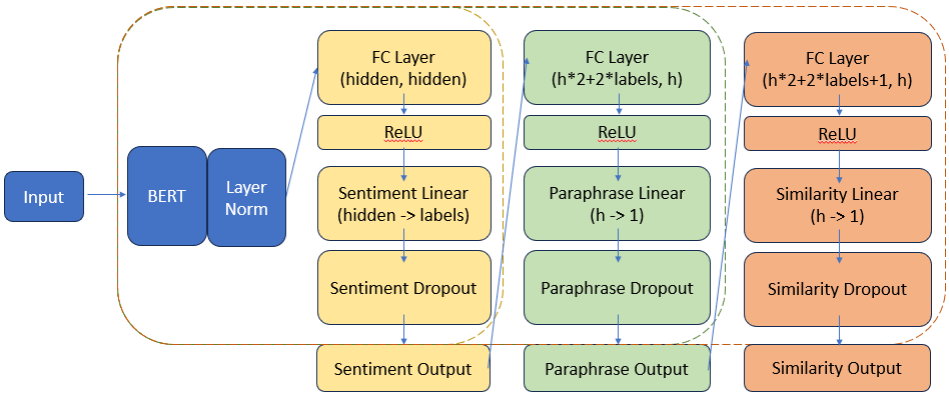
When I was doing preliminary research into my custom project I quickly came across the paper introducing Progressive Neural Networks (PNNs) (Rusu et al., 2022). This paper was aimed at tackling the catastrophic forgetting problem that occurs in transfer learning circumstances. It does this by instead of finetuning to a new task as an additional layer or unfreezing the last layer of the network, it creates lanes for each task. Where the initial task is trained like normal, but the subsequent tasks are trained with the outputs of the previous tasks layers as additional input.

4 Approach

In order to best judge the efficacy of any changes to vanilla finetuning, a baseline model was first created and trained. This model was built on an implementation of BERT as described in Devlin et al. (2019). Also, an efficient implementation of Adam optimizer was developed based on Kingma and Ba (2017). In this baseline model each task starts with the pooler output of BERT, then goes through its own sequence of fully connected layer, ReLU activation layer, then another final fully connected layer. After that the output is input to a loss function that is specific to the type of task. Sentiment analysis has 5 discrete categories so it uses cross entropy. Paraphrase is binary so it uses a sigmoid normalization and then binary cross entropy. Similarity correlation is a regression based output with unnormalized logits that can go between 0 and 5, so that is fed through an Mean Squared Error (MSE) loss function.

The final model ended up with the following architecture after many iterations of experiments. The pooler output of the BERT model is passed to through a LayerNorm layer with eps=1e-12. Each of the 3 tasks starts with this output. The sentiment task doesn't depend on any of the other tasks so it just has a sequence of fully-connected layer, relu activation, fully-connected layer with num_labels output, and then a task specific dropout layer. The paraphrase task has the sentiment task as a dependent. So it first feeds the output from BERT to the sentiment task for each batch of sentences. It freezes the weights of the sentiment layers when doing this. That was something I found was necessary during the experiments. It concatenates the BERT outputs with the sentiment outputs and passes this through its own sequence of fully-connected layer, relu activation, fully-connected layer with a binary output and a final dropout layer. The similarity task has the same basic structure. It freezes both the similarity and paraphrase layer weights and first passes the BERT output through those two tasks before doing its own sequence of layers.

The following is a diagram of the basic structure of this model:



5 Experiments

Since I was building my own extension to the default project instead of one that was preestablished, I wasn't exactly sure what was going to work or not. So, there was a lot of make a change, train, see if dev scores improved or if the training time increased too much, repeat. The main changes I made in my experiments were the following:

I started off with just passing the output of the sentiment task to the other two tasks. I didn't do any freezing of weights until the end of the experimenting. I talk later about how this made it very difficult for me to determine what exactly it was that caused the improvements seen in the final model. I then started passing the paraphrase output to the similarity task. I did this instead of the other direction for two reasons. The context made sense to me. I was still thinking in this simple to complex mindset and the paraphrase task was simpler in its goal than the similarity task. Also, the paraphrase task started out with a higher accuracy, so I thought the context given by it would be generally more useful. However, doing this increased the training time substantially until I started freezing the weights.

After getting this flow working I experimented with adding LayerNorm and task specific dropout. This did seem to have some improvement over the previous examples, but it was mixed. Probably, again, due to not freezing the weights of the imported tasks. It was now that I started figuring out how to freeze the weights and make that work with all the tasks and the training improved immediately. Steady progress was being had instead of the huge jumps and losses in accuracy I was getting between each epoch before.

5.1 Data

There are 3 tasks being trained for in this project and each is being trained on a different dataset. The Sentiment Analysis task is being trained on the Stanford Sentiment Treebank dataset (**?**). Paraphrase Detection is being trained on Quora's question pair dataset [1]. The Semantic Textual Similarity task is being trained on the SemEval STS Benchmark Dataset as listed in this paper **?**. The quora dataset has much more training data in it than the other two datasets. It has 17688 entries whereas the other two have 1068 and 755.

5.2 Evaluation method

The primary evaluation metric I used was the dev accuracy score. I averaged the scores for each of the 3 tasks and would save the best model during training. This was a good metric to be able to see whether or not the changes I made during my experiments were having a good effect on finetuning or not. Another evaluation metric I used was training time. When I began my experimenting I did not freeze the weights of the previous tasks when training a task that imported their outputs. This increased the training time by 400% but gave meager improvements to the actual dev accuracy score

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

as a result. This caused me to look into what was causing the drastic increase in training time and how to resolve that. So it was an important evaluation metric for me to use during my experiments.

5.3 Experimental details

A difficulty I had during this was the length of time it took to pretrain and finetune the model for each iteration of changes I made for the experiments. I did not get to make changes to hyperparameters as I would have liked since I was instead experimenting with the architecture of my model. Below is a table of hyperparameters I used for pretraining and finetuning.

	Pretrain	Finetune
Learning Rate	1e-3	1e-5
Epochs	10	10
Dropout Rate	.3	.3
Batch Size	8	8

Training Hyperparameters

5.4 Results

The results were were generally better than I expected. I was very pleased by the improvement in the similarity task. That really only occurred once I froze the weights. As soon as that began training I knew that was the correct way to go. The training times were better and the tasks steadily improved instead of bouncing around a lot and getting stuck. I was surprised by the improvement to the sentiment layer. I was expecting that to stay largely the same.

	Baseline	Shared Layer
Sentiment	.439	.488
Paraphrase	.819	.829
Similarity	.508	.591
Training Time	$\sim 8hrs$	$\sim 8hrs$

Test Accuracies and Total Training Time

6 Analysis

My model really only started working well once I began freezing the other tasks weights when I was training one of the downstream tasks. I believe this has to do with the gradients clobbering one another for those downstream tasks and the tasks not being able to optimize on their specific task. And since they weren't able to improve, the context they were providing to their downstream tasks was less worthwhile. With the other tasks weights frozen, each task's weights are only updated based on how well the task did at its specific goal, never some downstream goal that it has no context for.

I was also surprised by the increase to the sentiment analysis task. It is not gaining any context from the other two tasks. So only a few things change, the layer norm and the dropout layer as well as the better finetuning to the underlying BERT weights when the other tasks were performing better due to their increased context from the previous tasks.

The paraphrase task did not improve much. In hindsight this is also to be expected. Since this task had so much more data than the sentiment analysis task, it makes sense that the additional context provided by the sentiment analysis output wouldn't have moved the needle much for its own accuracy. Also, the sentiment analysis task was working with a sub 50% accuracy rate. The additional context provided by it would have come with a lot of noise.

The similarity task finished with the best improvement. This makes sense since it is taking in the additional context of the previous two tasks. The paraphrase task has such high accuracy, the context it provides would have been very beneficial to the similarity task. Although, I could imagine it hurting it for those examples where the similarity is rated around a 3 and the paraphrase task can only give it a binary of 0 or 1. The similarity task may be taking that context and overweighting the results towards 0 and 5, the extremes of its output.

7 Conclusion

Sharing the outputs of tasks with other tasks can increases the downstream task's accuracy. It relies on whether or not the context provided by the previous tasks makes sense for the downstream task. It made sense that sentiment analysis and paraphrase detection could help when determining if two sentences are similar. However, a flow the other direction would not have made sense at all. Also, when there is a large discrepancy in the training data sizes, the context given by certain tasks might not have as much impact as the one with more training data and higher accuracies.

This project required considerable iteration to find a result that worked well and improved over the baseline. This is a lot of time taken up waiting for the training cycle to complete. Once I determined that it was necessary to freeze and unfreeze the previous tasks weights, there just wasn't enough time left to rerun the previous experiments to compare with that worked out. There are many gaps in experiments that would be beneficial to pin down exactly what is contributing to the increase in accuracy. What if training is done only importing sentiment analysis and not paraphrase and vice versa. How does this impact the results of the similariy task? What if more was done to increase the accuracy of the sentiment analysis task, be it more pretraining or more data or other means. How would this impact the improvements gained by the downstream tasks. What if we didn't completely freeze the tasks weights during finetuning but just decreased the learning rate for those layers. What if some more sophisticated means were used to combine the outputs from the previous task with the inputs of the current tasks. Some multi-field attention method where the weights used to combine them into one representation could also be learned.

Overall, I showed that when the context makes sense, taking in the output of one task as part of the input of a subsequent task is valid method of increasing the improvement of those downstream tasks.

References

Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. 2022. MTRec: Multi-task learning over BERT for news recommendation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2663–2669, Dublin, Ireland. 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. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2022. Progressive neural networks.