BERT Finetuning Analysis

Stanford CS224N Default Project

Xueying Xie
Department of Computer Science
Stanford University
xueyingx@stanford.edu

Abstract

BERT is pretrained to be a generalist at language representation. It presents itself as a great stepping stone for investigations into increasing its accuracy for more specific tasks, like sentiment analysis, paraphrase detection, and semantic textual similarity. The goal is to experiment with different architectures for each task, loss functions, and fine-tuning techniques in order to build a high performing model across all three subtasks. Since the contributions were mostly in finetuning, the findings make it clear that without further pretraining where BERT gets direct access to task-specific data to update its parameters, finetuning's limitations require much more work to be comparable. The experiments were able to achieve improvements from baseline scores by almost three folds. These promising results demonstrate that further improvements can be easily attained with better finetuning or other techniques outside of this paper's scope.

1 Key Information to include

- Mentor: Sauren Khosla
- External Collaborators (if you have any): None
- Sharing project: None

2 Introduction

Language models have been at the forefront of natural language processing research for many years. With the advent of large pre-trained models such as BERT and GPT-3, researchers have been able to achieve state-of-the-art results on a wide range of language tasks. However, these pre-trained models are often not directly applicable to specific downstream tasks, and fine-tuning is necessary to achieve optimal performance.

Fine-tuning language models involves taking a pre-trained model and fine-tuning it on a specific task by training it on a task-specific dataset. This approach has shown significant improvements over traditional machine learning approaches that require building task-specific models from scratch. Fine-tuning language models have been successfully applied to various tasks such as sentiment analysis, named entity recognition, question answering, and text classification.

One challenge with fine-tuning pretrained models for multiple tasks is that it's hard to find a reliable representation overlap and may hurt each other in performance. It's not always clear where conflicts between tasks are occurring, but that there exists some sort of contention between tasks as they don't always increase in performance together. Instead some increase while others decrease. For the three tasks of interest explored in this paper, the tasks are relatively homogeneous, but some contention can be detected between the three when different architectures are mixed and matched between the three heads.

This paper aims to provide some insight into finetuning BERT for multitask learning on sentiment analysis, paraphrase prediction, and semantic textual similarity. The paper will also provide a

comprehensive analysis of the experimental results on three datasets – SST, QQP, and STS – to demonstrate the efficacy of fine-tuning pretrained models for multitask classification.

3 Related Work

Fine-tuning BERT (Bidirectional Encoder Representations from Transformers) has become a popular approach for many natural language processing (NLP) tasks. In this section, we provide an overview of some of the key works related to fine-tuning BERT.

The original BERT model [1] is a pre-trained transformer-based model that can be fine-tuned on specific downstream NLP tasks. Since then, there have been many studies on fine-tuning BERT for different NLP tasks. Sun et al [2] investigated text classification, while Bi et al [3] focused on news recommendation. Adaptations and augmentations were also proposed in the aforementioned works. Others include PAL [4] and data augmentation techniques [5].

For the downstream tasks of interest in this paper, Sentence BERT [6] uses siamese and triplet network structure to derive meaningful sentence embeddings. It is particular useful for sentence pairs and finding similar pairs of sentences.

4 Approach

The approach of this paper's BERT finetuning follows closely with that of BERT's section 3.2 [1]. For each task, each sentence are passed into BERT independently and finetuned together. Concatenation of sentences before BERT ingestion and after embedding generation are also experimented. Once embeddings were outputted from $BERT_{BASE}$ [1], they were fed into each downstream heads with the different architectures12. Different loss functions and parameter updates were experimented.

4.1 Architecture

The main effort of this project was put into creating finetuning architecture for each downstream task. Although the essence of each downstream task is a classification problem, the inputs and outputs of each dataset differs and may better fit different architectures with x as the embedding outputs from BERT. Figure 1 shows bi-encoder method, while Figure 2 shows cross-encoder method for STS.

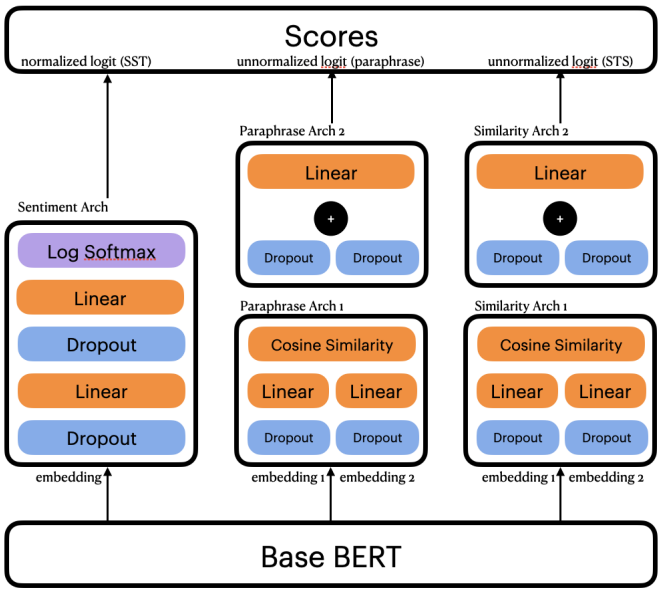

Figure 1: Basic Architectures for Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity.

Sentiment analysis represents a typical N-class (where $n > 2$) classification problem that defaults the architecture to some type of multi-layer perceptron model. Due to resource constraints, only 2 linear layers (1) were able to be trained.

$$\begin{aligned} h &= xA_h^T + b_h \\ y &= hA^T + b \\ s_i &= log(\frac{exp(y_i)}{\Sigma(exp(y_j))}), \end{aligned} \tag{1}$$

Paraphrase detection represents a typical binary classification (2) problem that outputs a single score that represents two classes in that $label_1 = p$, $label_2 = 1 - p$, where p could be a softmax score. On the other hand, it can also be a problem solved by some sort of similarity metric (3), where the threshold is the midpoint of the scores' value range.

$$s = \begin{bmatrix} x_1 & x_2 \end{bmatrix} A^T + b \tag{2}$$

$$\begin{aligned} xlin_1 &= x_1 A_1^T + b_1 \\ xlin_2 &= x_2 A_2^T + b_2 \\ s &= \frac{xlin_1 \cdot xlin_2}{max(||xlin_1||_2 \cdot max(||xlin_2||_2)} \end{aligned} \tag{3}$$

On the same vein, semantic textual similarity could also be seen as a classification problem. Its input and outputs are the same as paraphrase detection. If the STS dataset labels are seen as discretized values, there would be $(5 - 0)/0.2 = 25$ labels for every pair of sentences. Instead of classifying to 20 classes, the STS head architecture can still utilize linear classification or similarity metric scoring as described in equations (2) and (3).

Since STS is a slightly different problem than paraphrase detection, cross encoders may better represent the input pairs better. As recommended by BERT's section 3.2 [1], it claims that "encoding a concatenated text pair with self-attention effectively includes bidirectional cross attention between two sentences". This could be very intuitive as BERT see both sentences together, instead of separately and capture relations between them when generating embeddings. This would put the onus less on downstream architecture to perform similarity measurements.

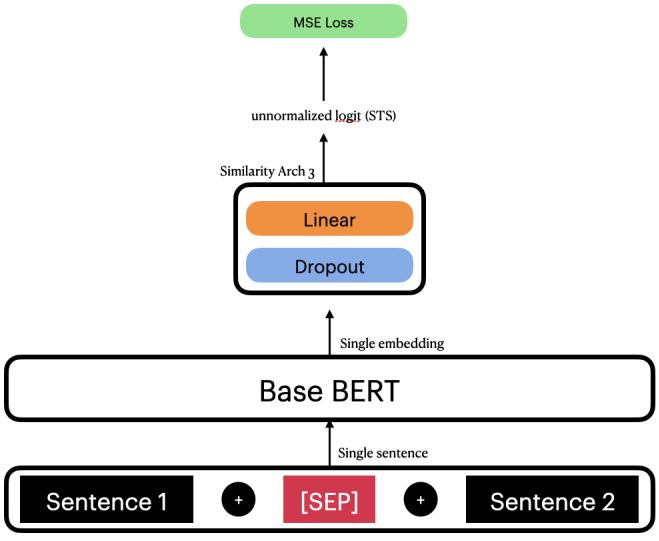

Figure 2: Alternative Architecture for Semantic Textual Similarity.

4.2 Parameter Updates

With three downstream tasks of different dataset sizes, there are a few different ways to update parameters3. As a pseudo-baseline, parameter updates can happen serially and independently between tasks with the summation of losses at the end of each epoch. Building on from that, if training is ran in a round-robin fashion, parameter updates can happen at every batch or at the end of all three batched tasks.

```
(a) serial-run

For batch in SST dataset:
     SST Loss
     Optimizer step

For batch in PPQ dataset:
     PPQ loss
     Optimizer step

For batch in STS dataset:
     STS loss
     Optimizer step
```

```
(b) round-robin-sum-loss

For batch in SST dataset:
     SST Loss

For batch in PPQ dataset:
     PPQ loss

For batch in STS dataset:
     STS loss

Loss = SST loss + PPQ loss + STS loss
Optimizer step
```

```
(c) round-robin-run

For batch in SST, PPQ, STS dataset:
     SST Loss
     Optimizer step

     PPQ loss
     Optimizer step

     STS loss
     Optimizer step
```

Figure 3: Optimizer update options.

4.3 Loss Functions and Scaling

Different loss functions can suit different problems. For typical classification problems, such as sentiment classification, cross entropy loss is the most standard objective to use. For binary classification problems, such as paraphrase prediction, binary cross entropy loss or regular cross entropy loss would be best. For semantic textual similarity, cross entropy loss is attempted to see if the problem space can be objectified as a classification problem. L1 and L2 losses are also experimented with to see if it is better objectified as a regression problem.

Experiments loss scaling were also conducted in order to see if it can aid in the dataset size imbalance. The idea is to give more weight to smaller datasets, so the larger datasets will not dominate for particular downstream tasks. In particular, losses for SST was scaled up about 15x per epoch, while lossees for STS was scaled up about 25x per epoch.

5 Experiments

5.1 Data

All datasets are provided by CS224N. Sentiment analysis task uses the Stanford Sentiment Treebank dataset. Paraphrase detection task uses the Quora Question Pairs dataset. Semantic textual similarity task uses the SemEval Semantic Textual Similarity dataset. An overview of the datasets are provided in Table 1.

Dataset	Inputs	Classes	Train	Dev	Test
SST	1 sentence	5	8544	1101	2210
QQP	2 sentences	2	141506	20215	40431
SemEval STS	2 sentences	6	6041	864	1726

Table 1. Statistics of three datasets used in multitask classifier investigation.

As described in the default project bert handout, SST dataset contains single sentence movie reviews that can be negative(0), somewhat negative(1), neutral(2), somewhat positive(3), or positive(4). The QQP dataset contains sentence pairs that are labeled as is paraphrase (1) and is not paraphrase (0). The STS dataset also contains sentence pairs that are labeled from 5 (same meaning) to 0 (not at all related).

5.2 Evaluation method

Evaluations follow what was given in the cs224n default bert project code base. It utilizes standard accuracy5 for sentiment classification and paraphrase detection and Pearson correlation4 for STS. In addition, confusion matrices are analyzed and helped debug during the experiments.

$$pearsoncoeff = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4}$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

5.3 Experimental details

All permutations of the different architecture, loss, and parameter stepping options in section Approach4 were experimented to find the best combination to train with the least contention and best performance.

With the base BERT model, default training parameters5.4 were used. The AdamW optimizer was employed across all experiments with constant hyperparameters of 1e-3 learning rate, 0.9 beta1, 0.999 beta2, 1e-6 epsilon, 0 weight decay, and enabled bias correction.

For extensions to the BERT model, some training hyperparameters tuned5.4 were inspired from the BERT[1] repository's example jupyter notebook.

	epoch	batch size	hidden dropout prob	lr
default params	10	8	0.3	1e-5
experimental params	10	32	0.4	2e-5

Table 2. Default and tuned model configurations.

5.4 Results

	SST accuracy	PPQ accuracy	STS correlation	Overall Accuracy
baseline	0.317	0.376	0.019	0.238
serial-run (param update baseline)	0.309	0.591	0.232	0.377
round-robin-run	0.519	0.684	0.323	0.509
round-robin-sum-loss	**0.536**	0.604	0.255	0.465
use-all-data	0.511	0.758	0.306	0.525
loss-scaling	0.508	0.600	0.348	0.485
SST-PARA1-STS1 (arch baseline)	0.519	0.684	0.323	0.509
SST-PARA2-STS1	0.514	0.707	0.290	0.504
SST-PARA1-STS2	0.503	0.482	0.313	0.433
SST-PARA2-STS2	0.517	0.661	0.321	0.500
STS-cross-encoder	0.484	**0.768**	**0.607**	**0.620**
param-bs32-dp0.3-lr1e-5	0.517	0.751	0.293	0.520
param-bs32-dp0.4-lr2e-5	0.505	0.740	0.309	0.516

Table 3. Performance of experiments on BERT extension dev set.

	SST accuracy	PPQ accuracy	STS correlation	Overall Accuracy
dev set leaderboard	0.484	0.768	0.607	0.620
test set leaderboard	0.500	0.772	0.614	0.628

Table 4. Performance of final BERT extension dev and test set.

Different loss functions were experimented with paraphrase prediction and textual similarity, as mentioned in Section 4.3. L1 and L2 losses for STS worked the best, while cross entropy worked slightly better for paraphrase detection. Once this was established, the remaining experiments coudl continue with more confidence.

The first three rows of results with entries "serial-run", "round-robin-run", and "round-robin-sum-loss" encompass the Approach section 4.3. Running the steps serially was clearly the worst option as each task dominantly updating first third of training, then second third, then last and not learning concurrently. Hence, the results were quite poor, but was a good starting point for trying different loss functions and minor architecture changes because it trained faster than the other approaches. Another approach of updating parameters experimented with summing up losses of each task before allowing the optimizer to take a step. This proved to be insignificant as the result was reducing 3 steps to 1 step. The results from the first three rows concluded that stepping more often improves performance.

Once the best parameter stepping frequency is established, the next four rows of results with entries "SST-PARA*-STS*" show results of the Approach section 4.1. PARA1 and STS1 corresponds to cosine similarity architecture, while PARA2 and STS2 corresponds to the architecture that concatenated inputs into a linear layer. The findings from this section seem to indicate that either architecture for the STS task works fine, but when trained together with PARA, the overall system prefers cosine similarity. The alternative architecture 2 proposed in section 4.1 boosted the performance of STS task more substantially than any other non-architectural changes.

Up until this point in the experiments, only a portion of the paraphrase dataset had been used in order to iterate experiments faster and balance the datasets so they're similar in size. The row named "use-all-data" round robin iterates through all datasets irrespective of length and brings back data imbalance towards paraphrase dataset as it has ≈ 20 times more examples than the other two datasets. The main potential implication was that once the smaller datasets trained, only the larger dataset will dominate the remaining training iterations. The results disproved that the imbalance was a major issue for overall accuracy. However, when loss was scaled up for STS and SST, SST performed the best as it became on par in terms of influence at each parameter update. Ultimately, more good quality data will improve overall accuracy of the system.

Lastly, a few hyper parameters were tuned to see its effect on the accuracy of each task alone and overall tasks together. Batch size, hidden dropout probability, and learning rate was increased. Given the results, no improvements were seen from the adjustments. This is not indicative of triviality of hyper parameters. The focus of this paper was not given to tuning, so many other permutations could have improved the accuracy, but just weren't experimented in this scope.

6 Analysis

In this section, some thoughts are shared regarding the main extensions experimented in this paper.

6.1 Hyperparameter and data limitations

Even though SST, QQP, and STS are quality datasets, the accuracy of PPQ over the other downstream tasks, especially STS is tremendous. Without pretraining, the most reliable way to make the model more robust to the STS task may be to have more data. Though not attempted in this paper, there was consideration of creating more samples in the STS dataset by generating new sentences by changing names and pronouns in existing samples. Otherwise, no amount of hyper parameter tuning could drastically change the results. It is evident that STS accuracies were slightly higher when PPQ dataset was down-sampled or its loss was scaled up in Table 3's "loss-scaling" result. It would have been ideal to concatenate more datasets together to create larger training set of at least 10-20k samples.

6.2 Architecture Shortcomings

With a single backbone architecture, the generalist option seems to taper in accuracy than training each task separately by potentially having different backbones that are more reliable for specific tasks.

With the alternative head2 for STS, there was substantial improvement in its performance. This further proves BERT's ability to bidirectionally cross encode. Although paraphrase detection performs quite well with cosine similarity, its score ultimately results in a binary classification, so any values above or below a threshold is regarded the same. Therefore, this downstream task is not as sensitive to scoring ranges as STS. STS scores are much more granular and therefore more sensitive to even slight variations in scores. Despite it's decent performance, paraphrase detection could be further

improved if the embeddings were further manipulated. the alternate architecture2 applied to STS could be applied to paraphrase detection as well. Though it may be an overkill to learn such deeper semantic relations, when the result is just a binary yes or no. Perhaps these embeddings can be further generalized by boiling down the embeddings into an average or some other smaller representation.

Though potentially outside the scope of this project definition, the time overhead and embedding generalization of using BERT could be mitigated using SBERT[6]. Generating sentence embeddings like that of SBERT could instill richer relations that could benefit sentence pair similarity tasks.

6.3 Pre-training Need

As alluded to in the abstract, fine-tuning on a base BERT model may not be sufficient. The model may experience some semblance of zero-shot or few-shot problems. With further pretraining, allowing the model to see the domain-specific dataset would help it learn some representations to aid it in downstream predictions of the same tasks. Without pretraining, there's a lot of uncertainty in how the original pretraining tasks (NSP and MLM) can transfer well into other NLP tasks.

Though the limitation outlined in the previous subsection is most relevant to the STS task, maybe joint training is just not as robust as training individual tasks separately to avoid contentions in gradients. Though this paper did not do a thorough investigation into supporting this statement, there is some evidence from the fluctuating STS results in Table 3 and over-fitting performance of SST seen from large discrepancies between its train and dev accuracy.

7 Conclusion

This project attempted to different fine-tuning methods for 3 similar downstream tasks that achieved almost 3x increase in performance from the baseline model, but much lower than SOTA models and the cs224n default project leaderboard statistics. The experiments outlined in this paper showed that its focus areas like loss function, parameter updates, and hyper parameter tuning has insignificant impact on the overall robustness of a multi-classifier system. Instead, more custom architectural changes in each individual task could better help leverage base BERT and avoid gradient contention between tasks. Lastly, for more reliable performance, further pre-training would help learn more appropriate parameters for specific downstream tasks without the uncertainty of representation overlap with pretrained tasks.

References

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

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

[3] Lifeng Shang Xin Jiang Qun Liu Hanfang Yang Qiwei Bi, Jian Li. Mtrec: Multi-task learning over bert for news recommendation. 2022.

[4] Iain Murray Asa Cooper Stickland. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. 2019.

[5] Luchen Tan Kun Xiong Ming Li Jimmy Lin Wei Yang, Yuqing Xie. Data augmentation for bert fine-tuning in open-domain question answering. 2019.

[6] Iryna Gurevych Nils Reimers. Sentence-bert: Sentence embeddings using siamese bert-networks. 2019.