Finetune minBERT for Multi-Tasks Learning

Stanford CS224N Default Project

Yingbo Li
Department of Computer Science
Stanford University
yingboli@stanford.edu

Abstract

In this default project, I explore and implement a variety of fine-tuning options to improve model accuracy on three different types of downstream tasks, sentiment classification (SST data), paraphrase detection (Quora data), and semantic textual (SemEval STS benchmark data). The goal of this work is to evaluate whether some successful solutions in the literature can improve BERT's multiple-task learning accuracy for the three above-mentioned tasks. The options I implemented included layer-wise learning rate decay (Sun et al., 2019), the SBERT architecture and the average of BERT embedding (Reimers and Gurevych, 2019), and the sum of loss functions (Bi et al., 2022). In addition, I also conducted some naive hyperparameter tuning, by experimenting a few different values of learning rate and number of epochs. Comparing with a baseline model that freezes the original BERT parameters but only pretrains the three final output layers, one for each task, I find that some of these options, such as SBERT architecture and the average of BERT embedding, can greatly improve the model accuracy, while some others, such as layer-wise learning rate decay and sum of loss functions do not seem to make a difference or more worsen the performance. In addition, not surprisingly, the choice of learning rate matters a lot, and increasing the number of epochs is helpful.

1 Key Information to include

- Mentor: Yuan Gao
- External Collaborators (if you have any): None
- Sharing project: None

2 Introduction

Trained on huge corpus, large language models contains huge amount of information and knowledge about natural language, so that using one single model on multiple downstream tasks of different types become feasible. In this course project, based on the base BERT model, I evaluated several fine-tuning ideas that are proven success in the literature, in order to improve model accuracy on three different types of classification tasks, including sentiment classification (SST data), paraphrase detection (Quora data), and semantic textual (SemEval STS benchmark data). I notice the following challenges and will address them in my project.

Different natures of the downstream tasks: in the sentiment classification, each observation is a single sentence and the goal is to learn its sentiment. In contrast, for the paraphrase and similarity tasks, each observation (data point) contains a pair of two sentences, and the goals are to tell if they are similar. In other word, understanding each sentence along may not be sufficient, comparing the pairwise difference may also be critical. Therefore, instead of simply just concatenate the BERT outputs of the two sentences, I implement the SBERT structure in Reimers and Gurevych (2019), which also include the difference between the BERT outputs of the two sentences. This approach shows improvements on the paraphrase and similarity tasks.

Choices of sentence level BERT embedding: since in the BERT model, each token is a word or sub-word, so for the three downstream tasks that are all on the sentence level, some pooling, i.e. from sub-word to sentence aggregation, may be needed. Here, I evaluate two options, BERT output on the beginning of sentence `[CLS]` token, and also the averaging among the non-padding BERT tokens. Note that the later has been shown superior in Reimers and Gurevych (2019), and that is consistent with my finding here.

Different learning rate for different BERT layers: the base BERT model contains 12 layers of transformers blocks. The general belief is that the lower layers capture more basics about language while higher layers may contains more sophisticated information. Hence, the higher layers may need larger learning rate in the fine-tuning process while the lower layers may need smaller learning rate, or may even be frozen. For example, Sun et al. (2019) experimented only fine-tuning the last one or four layers of the BERT model, and report comparable accuracy with fine-tuning all 12 layers, for classification tasks. The same paper also evaluated layer-wise learning rate decay and shown superior performance. In this work, I also implemented layer-wise learning rate decay; however, the benefit seems to be unclear.

Combining different losses from the three different tasks: it is not surprising that the three downstream tasks are not perfectly consistent with each other. In the training process, a gradient update step that improve the accuracy of one task may sometimes hurt the accuracy of the other one or two. To alleviate this problem, I explore using the sum of three loss functions as the objective function, as proposed in Bi et al. (2022), however, the results are far from promising.

Hyperparameter tuning: in general the choice of hyperparameters can be critical for the success of deep learning models. However, since each model training (pretraining or fine-tuning, with descent amount of epochs) can easily take several hours of GPU time, it is very expensive to conduct sophisticated hyperparameter tuning (e.g., using some Bayesian optimization algorithm such as TPE in Optuna). Therefore, I have evaluated the two most important hyperparameters, learning rate and number of epochs, at a few different values. My findings are that choice of learning rate can be very important, and increasing the number of epochs may often lead to higher accuracy.

3 Related Work

The base BERT model (Devlin et al., 2018) that we study here contains 12 layers of multi-headed transformer blocks (Vaswani et al., 2017). With hidden size 768, and 12 heads in each layer, it has in total 110M parameters. It has been shown to achieve satisfying performance in a lot of tasks, by only fine-tuning the pre-trained BERT model with an additional final output layer.

Sun et al. (2019) investigate multiple approaches to improve BERT fine-tuning performance for classification tasks. Their recommendations include: handling long text via truncation, treating different BERT layers through layer-wise learning rate decay, and fine-tuning BERT on more training data by including extra datasets with similar types of tasks.

For sentence level tasks, Reimers and Gurevych (2019) paper proposed modifications of BERT structure for sentence-pair tasks that can derive semantically meaningful sentence embeddings. In particular, in their SBERT for classification structure, they show via empirical studies on semantic textual similarity tasks that (1) for the pooling step that converts the sub-word level BERT token to a sentence embedding, averaging the BERT tokens outperforms directly using the `[CLS]` token, and (2) for concatenating not only the two sentence embeddings themselves but also the absolute values of their differences achieves ideal performance.

4 Approach

4.1 Baseline: SBERT for paraphrase and similarity

To train the BERT model for three different types of downstream tasks, namely, sentiment classification on SST data, paraphrase detection on the Quora data, and semantic textual similarity on SemEval data, I update the training function to incorporate the losses from all three tasks in the following naive way: in each epoch, I perform

1. AdamW steps on the entire SST data in batches, using cross-entropy loss on the 5 target classes.
2. AdamW steps on the entire Quora data in batches, using binary cross-entropy loss with the sigmoid logits. Here, I followed the SBERT architecture for classification (Reimers and Gurevych, 2019), that is to concatenate the BERT final layer's output hidden states for sentence 1 u, sentence 2 v, and their absolute difference $|u - v|$ and then feed it to the final linear layer.
3. AdamW steps on the entire SemEval data in batches, using the mean squared error loss for regression. Although this is a regression task, the similarity/difference between the two sentences are critical for this task. Hence, similarly to the paraphrase task on Quora data, the hidden states I pass to the final linear layer is also $(u, v, |u - v|)$.

I have three variants of baseline results. The first one is to just pre-train the final output linear layers for each task with the original BERT parameters frozen at their official per-trained values. The second baseline is to fine-tune all parameters of the model, starting from the values in the original BERT parameters and randomized values for the output layers. The third baseline models is to fine-tune based on the first pre-trained model here.

During the training of the baseline models, I notice that the accuracy of sentiment classification task decreased a lot after the training of the semantic textual similarity task. So in order to reduce this accuracy drop, for all the following approaches, I adjust the order of the tasks in each training epoch, from (sentiment, paraphrase, similarity) to (paraphrase, similarity, sentiment). Unfortunately, this modification does not seems to help.

Results of the three baseline models on the dev datasets are listed in Table 2.

4.2 Pooling of BERT token to sentence level

In all above-mentioned baseline models, the sentence level embeddings are the BERT output for the start of sentence `[CLS]` token. It may make sense to do so because BERT is a bi-directional encoder model, and thus it is reasonable to assume that the `[CLS]` token may contain the information of the entire sentence. On the other hand, since the averaging of BERT embedding pooling approach has been shown success in Reimers and Gurevych (2019), I also implement this version to in two different ways. The first is to average BERT output on all three downstream tasks (see Table 3), and the second is to use [CLS] token for sentiment classification, and averages of BERT output on the paraphrase and similarity tasks (see Table 4).

4.3 Layer-wise learning rate decay

Denote the learning rate by l and the layer-wise learning rate decay rate as ξ. To implement the layer-wise learning rate decay, I let the learning rate of the last i-th layer of transformer block as

$$l^i = l \cdot \xi^{i-1}, \quad i = 1, 2, \ldots, 12$$

Note that the learning rate of the final output layers are set to l. Following the comparison of difference value of ξ in Sun et al. (2019), I set $\xi = 0.95$.

Table 5 and 6 contain my comparison of using layer-wise learning rate decay.

4.4 Sum of three loss functions

During the baseline training, I notice that an update step for the semantic textural similarity task often hurts the accuracy of the sentiment classification task, and vice versa. Since in the baseline training, I update the same set of parameters of the BERT model for the three downstream tasks in a one-task after another sequential way, I suspect that parameters may go too far in the direction of one task before the they are trained on the next task. Therefore, inspired by (Bi et al., 2022), I experimented a more natural way to combining the information in the three tasks, i.e., adding the loss functions from the three tasks up in each gradient update step, with equal number of random samples drawn

	Sentiment SST	Paraphrase Quora	Similarity SemEval STS
Train	8,544	141,506	6,041
Dev	1,101	20,215	864
Test	2,210	40.431	1,726

Table 1: Sample sizes of the train, dev, test data of each task.

from each task's training dataset (which equals to the batch size parameter). In other word, the loss function become the sum of the cross-entropy losses for the sentiment classification and paraphrase detection tasks, and the mean squared error loss for the semantic textural similarity task. Note that the sample sizes of the three training datasets are very different (see Table 1). In order to obtain the same number of training data points for all three tasks in each updating step, the smaller datasets are reshuffled and reused for many more times.

4.5 Different learning rates and more numbers of epochs

To understand the importance of the choice of certain hyperparameters, I evaluated some of the above mentioned methods in a few different learning rates (for fine-tuning, 10^{-5} and 2×10^{-5}) and number of epochs (20, 100).

5 Experiments

5.1 Data

For multi-task learning, three different types of tasks and their datasets are provided. Note that the sample size across the three tasks are very imbalanced (see Table 1). For the training sets, the paraphrase detection task (Quora data) has 15 to 20 fold more data than the sentiment classification task (SST data) and the semantic textural similarity task (SemEval STS data).

5.2 Evaluation method

The evaluation metric for sentiment analysis and paraphrase detection tasks is the accuracy rate, and evaluation metric for the semantic textual task is the Pearson correlation between predicted and actual similarities.

The overall scores are computed as the average of the three task metrics, with the Pearson Correlation been normalized first from the range $[-1, 1]$ to $[0, 1]$.

5.3 Experimental details

Unless mentioned otherwise, the default model specifications are

- Learning rate: 10^{-3} for pre-training and 10^{-5} for fine-tuning.
- Number of epochs: 20
- For the experiment where the layer-wise learning rate decay is used, the decay rate is set to $\xi = 0.95$, which is shown the best choice among a grid in Sun et al. (2019).
- Batch size is fixed at 32

After each epoch, if the overall score on dev datasets are improved, the model is saved.

5.4 Results

5.4.1 Baseline: SBERT for paraphrase and similarity

Among the three baseline models, not surprisingly, first pre-training the output layer, then fine-tuning the entire model achieves the highest accuracy for all individual tasks and also the overall score (see Table 2). It's worth mentioning that this version even achieves the highest accuracy of 0.526 for the SST sentiment classification task in this entire project, outperforming all following experiments.

	Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Similarity Pearson Corr.
Pre-train output layer	0.606	0.400	0.721	0.393
Fine-tune from original BERT	0.737	0.514	0.855	0.684
Fine-tune from pre-trained output layer	0.767	0.526	0.869	0.811

Table 2: Baseline results on the dev datasets. BERT output on the `[CLS]` token are used as the sentence level embedding. For the paraphrase and similarity tasks, SBERT structure is used, i.e., concatenation of two sentences' BERT outputs u and v, and also their absolute difference $|u - v|$.

	Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Similarity Pearson Corr.
Pre-train output layers, $\texttt{lr} = \texttt{10}^{-3}$	0.688	0.461	0.756	0.693
Fine-tune, 20 epochs, $\texttt{lr} = \texttt{10}^{-5}$	0.772	0.511	0.882	0.846
Fine-tune, 100 epochs, $\texttt{lr} = \texttt{10}^{-5}$	0.770	0.514	0.871	0.851
Fine-tune, 20 epochs, $\texttt{lr} = 2 \times \texttt{10}^{-5}$	0.762	0.493	0.875	0.839

Table 3: Pooling of BERT token results on the dev datasets. Average BERT output on the non-padded tokens are used as the sentence level embedding for all three tasks.

5.4.2 Pooling of BERT token to sentence level

Table 3 suggests that averaging of BERT tokens for all three tasks outperforms using `[CLS]`, for two of the tasks, paraphrase detection and semantic textural similarity. For the sentiment classification task, since averaging BERT output yields worse accuracy than using `[CLS]` token, I switched back and show results in Table 4. In this table, the accuracy for sentiment seems to slightly improve, but those for the other two tasks sometimes slightly worsen. Overall, I think the best trials among the two different versions are pretty comparable, with the highest overall score being 0.772 vs 0.773 on the dev datasets.

5.4.3 Layer-wise learning rate decay

Layer-wise learning rate decay does not seem to help improve the model accuracy. In Table 5, where the averaging of BERT outputs are used for all three tasks, using layer-wise learning rate decay hurts the performance. In Table 6, where BERT output on the `[CLS]` token is used for sentiment classification task and average BERT output on the non-padded tokens are used for paraphrase and similarity tasks, layer-wise learning rate decay does not make a difference.

5.4.4 Sum of three loss functions

Table 7 suggests that, surprisingly, switching to using sum of three loss functions as the multi-task learning loss greatly hurts the performance. This may be due to that tasks with smaller training sizes, sentiment classification and semantic textural similarity, are reused for way too many times than that of the paraphrase task. In this way, when looping through the paraphrase data for 20 times (20 epochs), the other two datasets may be looped 200 or even 300 times.

	Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Similarity Pearson Corr.
Fine-tune, 20 epochs	0.770	0.507	0.878	0.850
Fine-tune, 100 epochs	0.773	0.518	0.876	0.849

Table 4: Pooling of BERT token results on the dev datasets. For sentiment classification, BERT output on the `[CLS]` token is used. For paraphrase and similarity tasks, average BERT output on the non-padded tokens and the SBERT structure are used. $\texttt{lr} = \texttt{10}^{-5}$.

ξ	`lr`	Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Similarity Pearson Corr.
1	10^{-5}	0.772	0.511	0.882	0.846
0.95	10^{-5}	0.769	0.500	0.882	0.845
1	2×10^{-5}	0.762	0.493	0.875	0.839
0.95	2×10^{-5}	0.641	0.126	0.874	0.847

Table 5: Fine-tuning with layer-wise learning rate decay on the dev datasets (note that $\xi = 1$ means not layer-wise learning rate decay). Average BERT output on the non-padded tokens are used as the sentence level embedding for all three tasks.

ξ	`lr`	Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Similarity Pearson Corr.
1	10^{-5}	0.770	0.507	0.878	0.850
0.95	10^{-5}	0.770	0.507	0.878	0.850
0.95	2×10^{-5}	0.770	0.504	0.876	0.859

Table 6: Fine-tuning with layer-wise learning rate decay on the dev datasets (note that $\xi = 1$ means not layer-wise learning rate decay). For sentiment classification, BERT output on the `[CLS]` token is used. For paraphrase and similarity tasks, average BERT output on the non-padded tokens and the SBERT structure are used.

5.4.5 Different learning rates and more numbers of epochs

For the two fine-tuning learning rate choices, Tables 3, 5, 6 consistently indicate that 10^{-5} outperforms 2×10^{-5}.

For number of epochs, although in Table 3, 20 epochs outperforms 100 epochs for the dev datasets, in all other scenarios, for both dev datasets and the test datasets, 100 epochs is superior (see Table 4, 8).

6 Analysis

Base on fine-tuning performance on both the dev and test dataset, some successful experiments are:

- Using average BERT tokens on paraphrase detection and semantic textural similarity tasks
- Using more number of epochs
- Using learning rate 10^{-5}

Also, learning from some unsuccessful experiments are:

- Layer-wise learning rate decay with rate 0.95: usually hurts performance
- Sum of three loss functions: my current naive implementation hurts performance. It may due to the very difference number of samples in the training datasets across the three tasks. Also, the three loss function may be of different scale of values, due to the difference of the softmax vs regression loss functions.

ξ	Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Similarity Pearson Corr.
1	0.759	0.498	0.878	0.803
0.95	0.759	0.498	0.878	0.803

Table 7: Fine-tuning trained on sum of the three loss functions. `lr` $= 10^{-5}$. For sentiment classification, BERT output on the `[CLS]` token is used. For paraphrase and similarity tasks, average BERT output on the non-padded tokens and the SBERT structure are used.

Epochs	Overall Score	Sentiment Accuracy	Paraphrase Accuracy	Similarity Pearson Corr.
Average BERT tokens for all three tasks				
20	0.764	0.493	0.880	0.837
100	0.768	0.510	0.872	0.843
[CLS] token for sentiment, and average token for paraphrase and similarity				
100	0.770	0.512	0.875	0.846

Table 8: Fine-tuning results on the **test datasets**. No layer-wise learning rate decay, and `lr` $=$ `10`$^{-5}$.

7 Conclusion

This project greatly helps me to gain more hands-on experience with fine-tuning of large language models. The experience of exploring and experimenting different options is an exciting and rewarding journal for my development. As there always exists some level of randomness in the training process, my above qualitative evaluations might need more data points to verify. Also, as multiple guest lecturers pointed out, a lot of times no one know why (something works or not).

For some future work, I think the sum of loss functions approach worth more trials, maybe with a better designs. Also, due to time limit, there are other options I think would be worth trying, including gradient surgery (Yu et al., 2020), and training on more datasets of similar types of tasks (Sun et al., 2019).

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18*, pages 194–206. Springer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.