Efficient Multi-Task MinBERT for Three Default Tasks and Question Answering

Stanford CS224N Default Project

Rachel Yang
Department of Computer Science
Stanford University
ry2277@stanford.edu

Fanglin Lu
Department of Computer Science
Stanford University
luf@stanford.edu

Gerardus de Bruijn
Department of Computer Science
Stanford University
gdebruyn@stanford.edu

Mentor: Heidi Zhang

Abstract

In this project, we apply minBERT to the three default project tasks (sentiment classification, paraphrase detection and semantic textual similarity) and a fourth task (question answering[1] using SQuAD 2.0).

We tried three approaches to improve model accuracy beyond the per-task fine-tuning baseline: Siamese BERT cosine similarity, Siamese BERT vector pooling, and Cross Attention. None of these approaches had accuracy exceeding that of the per-task fine-tuning baseline; per-task fine-tuning achieved the best accuracy score among all the approaches we tried (12th out of 131 submissions on the TEST leaderboard, as of Sunday Mar 17, 6:30pm PST).

We also care about model efficiency (in terms of reduced memory footprint and training time). We tried two approaches to improve model efficiency beyond the per-task fine-tuning baseline, without sacrificing accuracy: Multi-task fine-tuning a single BERT model using roundrobin training with annealed sampling, and multi-task LoRA (Low Rank Adaptation). Multi-task fine-tuning a single BERT model using roundrobin training with annealed sampling got scores very close to the per-task fine-tuning results, and would have ranked 15th on the TEST leaderboard. We also found that roundrobin training with annealed sampling negatively affected question answering and sentiment classification accuracy more than it negatively affected the accuracy of the three default project tasks.

1 Introduction

When fine-tuning a model for multiple tasks, there are various considerations that need to be taken into account. Which option provides the best overall performance (accuracy) for all the tasks? Which option is the most efficient in using memory and storage (which is especially important if those resources are limited, like on a mobile phone)? Which option requires less training time? A

[0]Team contributions: Guy implemented or contributed to: minBERT, AdamW optimizer, Siamese BERT cosine similarity/vector pooling/cross attention, round-robin training with annealed sampling, LoRA, SQuAD task. Rachel implemented or contributed to: AdamW optimizer, per-task fine-tuning baseline for the three default tasks, LoRA, SQuAD task. Fanglin implemented most of the SQuAD task.

[1]In question answering using SQuAD 2.0, the model is given a question and a paragraph, and the model must find the answer within the paragraph, or predict that the answer is not present.

model that is efficiently being trained requires fewer resources (like hogging GPUs on cloud service providers) and has less environmental impact.

To improve accuracy, we try tweaking the outer layers on top of the BERT model with various strategies, focusing on the sentence comparison tasks: paraphrase and text similarity. The BERT encoder model is based on the original BERT paper Devlin et al. (2018). As in the BERT paper, as the default comparison strategy we send two sentences concatenated as a single sequence into the encoder, separated by the [SEP] separator token. This is followed by a task-specific linear layer. We look at three other strategies to compare two sentences where we use the "Siamese BERT" model, as described in paper Reimers and Gurevych (2019). We first send the two sentences through a BERT model separately, and then compare the two outputs using the following approaches to determine the similarity: Cross Attention, Vector Pooling and Cosine Similarity.

To improve model efficiency, we try options to efficiently fine tune a single BERT model for all three tasks. This includes round robin training with annealed sampling from the three datasets (Stickland and Murray, 2019), and Parameter Efficient Fine Tuning (PEFT) using multi-task LoRA (Low Rank Adaptation) (Hu et al., 2021) where we inject a limited number of trainable parameters for each task into the BERT Layers of a single BERT model. Upon training and inference, the trainable LoRA parameters for the specific task are pooled with the existing BERT parameters that are kept frozen.

At the end, we also applied the minBERT to Question Answering with SQuAD 2.0 (Rajpurkar et al., 2018), and compared SQuAD 2.0 performance between a per-task fine-tuned model and a model using round robin training with annealed sampling.

2 Related Work

This project aims to replicate the BERT model approaches from the following works, and compare each approach's performance using the same underlying BERT-BASE model, to improve our own and readers' understanding about accuracy vs efficiency tradeoffs when using BERT.

As our baseline, we replicate the per-task fine-tuning approaches for classification tasks and SQuAD 2.0 described in Devlin et al. (2018).

Stickland and Murray (2019) explores how to adapt a single large base model to work with multiple tasks, focusing on multi-task learning on several natural language understanding tasks. From this paper we take two concepts: Parameter Efficient Fine Tuning (PEFT), and round robin annealed sampling. In the end, for PEFT, rather than using projected attention layers (PAL) as described in Stickland and Murray (2019), we have decided to use LoRA, from (Hu et al., 2021): Add trainable low-rank layers within minBERT while freezing pre-trained weights. We pivoted from PALs to LoRA because Hu et al. (2021) has more citations and has been referenced more in our own machine learning work experience.

Additionally, as explained in the "Approach" section below, we replicate the cross-attention approach in Vaswani et al. (2017), and the Siamese BERT approach in Reimers and Gurevych (2019).

3 Approach

3.1 Overview

In this section, we describe in more detail the various approaches we have used. Some of those can be used in combination (not exclusive). We describe three approaches for fine-tuning: keep BERT parameters frozen and only train the parameters for the task-specific outer layers, fine-tune a BERT model for each task, or use parameter efficient fine-tuning of a single BERT model.

We obtained the highest accuracy score using the per-task finetuning baseline described in Devlin et al. (2018). Our model had a minBERT model encoder instance for each of the four tasks that we fine-tuned. For tasks that required comparing two sentences (paraphrase detection, textual similarity, and SQuAD 2.0), we replicated the BERT paper's approach of concatenating the two sentences and providing the concatenated sentence as input to the BERT model. Section "3.2 Fine-tuning BERT" of Devlin et al. (2018) describes the model architecture for the three default project classifcation tasks.

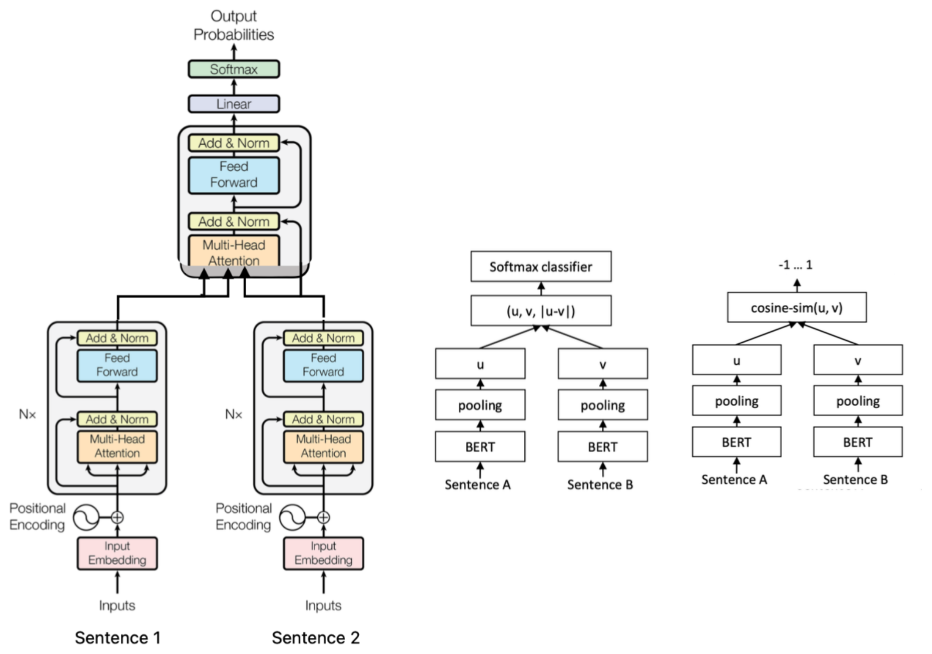


Figure 1: Cross attention, Vector Pooling and Cosine Similarity between outputs of two BERT encoders.

We obtained the lowest scores by freezing the pre-trained minBERT model parameters, and only fine-tuning the outer layers for each task. Other approaches are explained in the following sections.

3.2 Cross Attention

Cross attention is a strategy that we can use to compare two sentences. This approach is applicable for the paraphrase and the sentence similarity tasks (and SQuAD 2.0, but we did not test this approach on SQuAD 2.0). With cross-attention, we first send each sentence through the BERT model separately, and then use cross-attention between the last hidden states of the two outputs. The shortest sequence (sentence) is padded to make sure the two have the same length. With multi-head attention we then have $head_i = Attention(Q_{s1}W_{Q_i}, K_{s1}W_{K\,i}, V_{s2}W_{V\,i})$ where $s1$ refers to hidden states coming from the outputs from sentence 1, and $s2$ to hidden states coming from outputs from sentence 2.

This is a variation of the encoder-decoder cross attention as described in Vaswani et al. (2017). We use cross attention in the encoder. In our cross attention, the value weights are applied to the second sequence (sentence), going in one direction. [2] See diagram 1 for a visualization of cross attention.

3.2.1 Siamese BERT

There are two other options to compare two sentences based on the outputs of two parallel BERT encoders as described in paper Reimers and Gurevych (2019). In the vector concatenation option, we take the pooler output for each sentence, and pool those together before applying the softmax classifier. In paper (Reimers and Gurevych (2019)) they describe concatenating the two pooler outputs and their differences.

In the cosine similarity option, we calculate the cosine similarity between the pooler output for each sentence. It produces a scalar value in the range [-1, 1], where -1 means the two sentences are

[2]Rather than applying cross-attention again and again after each token is generated in the decoder, we only apply cross attention once per sentence pair. Another difference with the decoder is that the second sentence/sequence is completely "seen" (vs. "unseen" while generating the output sequence). That means the attention mask that we apply/add before the softmax is used to zero out the probabilities for the tokens only after the length of the second sentence.

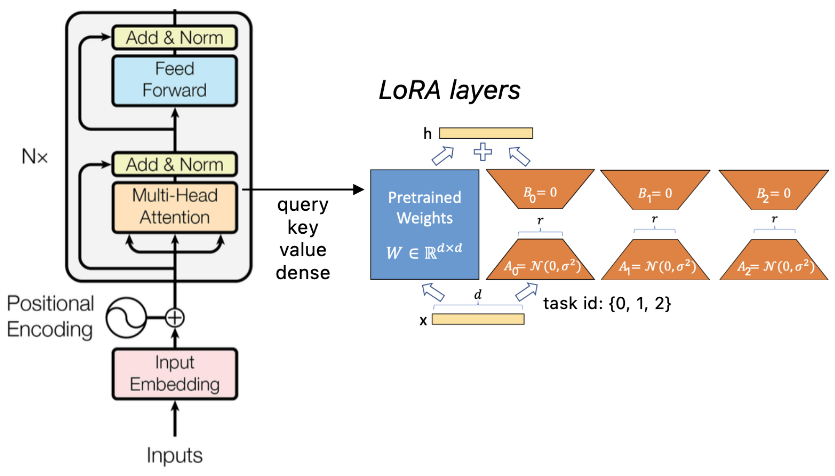


Figure 2: LoRA Layers

completely opposite, and 1 means the two sentences are exactly the same. In the prediction function we extrapolate/scale from [-1, 1] to favor the class label or similarity score that is the most likely.

See appendix A for details on the formula.

See diagram 1 for a depiction of Vector Pooling and Cosine similarity.

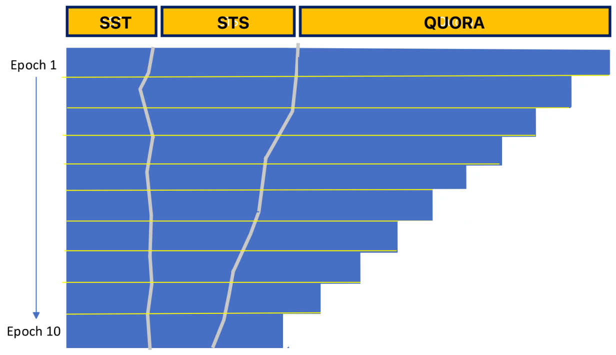
3.3 Multi-task LoRA

With LoRA (Low-Rank Adaptation) (Hu et al., 2021), we add trainable low-rank layers within minBERT while freezing pre-trained weights. This reduces training time because only a limited number of parameters have to be updated. Model storage cost is also low because only a single BERT encoder is used, while the size of the additional parameters is limited. Our initial implementation of LoRA for a single task was based on the BERT-LoRA-TensorRT (alexriggio) repository on GitHub. After that, we enhanced our implementation with the ability to train a single BERT model with LoRA weights for multiple tasks simultaneously, as follows: Let's refer to the key, query, value and attention dense layer in a BERT layer's self-attention block as W_k, W_q, W_v, and W_o respectively. Each W_k, W_q, W_v, and W_o matrix in each BERT self-attention layer has a LoRA layer attached to it. Each LoRA layer has one pair of A and B matrices for each task, where each A and B matrix is indexed by the task id; task id is 0 for sentiment, 1 for paraphrase, and 2 for semantic textual similarity. A_i and B_i refer to the A and B matrices for task i. In each LoRA layer, the A_i and B_i matrices corresponding to taskid i are multiplied and added to the layer weight matrix W (where W may be W_k, W_q, W_v, or W_o). Concretely, for input x and taskid i, the output of a BERT layer's attention key computation would be $W_k x + \alpha B_i A_i x$ instead of $W_k x$. W is the pre-trained BERT matrix in $\mathbb{R}^{d \times k}$. B is a matrix in $\mathbb{R}^{d \times r}$. A is a matrix in $\mathbb{R}^{r \times k}$. So, BA is the same shape as W. r is the rank of matrix BA and is a hyper parameter when training the model. A typical value for r would be 16. Another hyper parameter, α, is for scaling, used to impact of the adaptation. In our BERT model, both d and k are the size of the hidden layer, 768. See diagram 2 how LoRA layers are constructed.

Because $r \ll min(d, k)$ (our BERT model has hidden size 768), the additional memory and storage required for the A and B matrices is very limited, and fewer parameters need to be updated during training. The size with each additional task increases with a factor $\frac{2r}{hiddensize}$. So, with $hiddensize$ = 768 and rank r = 16 that means only about a 4% model size increase per additional task.

3.4 Round Robin with Annealed Sampling

In one of our first experiments, we fine-tuned a single BERT model sequentially on the three tasks. During training, each time after completing a task, the model performs well on that particular task, similar to the upper baseline per-task fine tuning. However, when training continues on the next task, the model "unlearns" some of what it was fine-tuned on for the previous tasks and the performance on



$$p_i \propto N_i^{\alpha}$$

$$\alpha = 1 - 0.8\frac{e-1}{E-1}$$

Figure 3: Annealed Sampling: at each epoch e, determine sample probability p_i for task i, relative to task dataset size N_i adjusted by α. α changes with each epoch e. E is total number of epochs.

those tasks decreases. A solution for that is round robin training, where with each step we take one batch from each dataset so that we simultaneously train all three tasks. Downside is that the number of steps is limited to the size of the smallest dataset (in this case the SST dataset for sentiment) and we do not take advantage of the amount of data from the larger data set (the Quora for paraphrase). With annealed sampling, as described in Stickland and Murray (2019), we start the first epoch sampling from each dataset according to the size of that dataset. Gradually over the epochs we reduce the samples from the largest data set until at the last epoch where a similar number of samples is provided from each dataset.

3.5 Question Answering

We applied minBERT to question answering using SQuAD 2.0 as an additional extension task, in which the model is given a question and a paragraph, and the model must find the answer within the paragraph, or predict that the answer is not present. Section "4.2 SQuAD v2.0" of Devlin et al. (2018) describes the model architecture for the SQuAD 2.0 question answering task, which we replicated. In summary, we introduce two parameters on top of the minBERT model, a start vector $\mathbf{S}$ and an end vector $\mathbf{E}$. We provide the concatenated question and passage as input to the BERT model, and calculate the probability of token i being the start of the answer span by calculating the dot product between $\mathbf{S}$ and the token i's embedding in the final BERT layer (T_i), followed by a softmax over the CLS token and all the tokens in the paragraph: $P_i = \frac{\exp(\mathbf{S}T_i)}{\sum_j \exp(\mathbf{S}T_i)}$. An analogous dot product and softmax with $\mathbf{E}$ are used to calculate the probability of word j being the end of the answer span. In cases where no answer is present in the passage, the model is expected to predict with high probability that $i = 0$ and $j = 0$, i.e. both i and j are set to the final BERT layer's CLS token. Like in the paper, the answer span is chosen to be either $(i = 0, j = 0)$ (if no answer is present in the paragraph), or the i and j that maximize $\mathbf{S}^{\top}T_i + \mathbf{E}^{\top}T_j$ where $j >= i$, and i and j are both paragraph tokens with index greater than 0 (if an answer is present in the paragraph). The paper predicts a non-null answer when $s_{\hat{i,j};i,j\neq 0} > s_{i=0,j=0} + \tau$, where the threshold τ is seleted on the dev set to maximize F1, but we simply predict a non-null answer when $s_{\hat{i,j};i,j\neq 0} > s_{i=0,j=0}$. This is because we did not have time to train a good value for the threshold τ.

For our SQuAD 2.0 experiments, we used a learning rate of 1e-5 and batch size of 8, just as with the other per-task fine-tuning and round robin experiments.

4 Experiments

4.1 Data

We have used the provided Stanford Sentiment Treebank (SST) dataset for sentiment classification, Quora Question Pairs (QQP) dataset for paraphrase detection, and SemEval STS for text similarity.

For the question answering task, we used SQuAD 2.0 data. The SQuAD 2.0 dataset extends the SQuAD 1.1 dataset by allowing for the possiblity that no short answer exists in the provided paragraph, making the problem more realistic.

4.2 Evaluation method

For sentiment classification and paraphrase detection we use accuracy (number of correct predictions divided by the total). Because sentence similarity returns a number on a scale between 0.0 and 5.0, we take the Pearson correlation coefficient, which measures the covariance between the predicted result and the actual label.

For question answering, we evaluate exact match and F1 score of the predicted answer span's BERT tokens decoded to string, compared to the reference answer span's BERT tokens decoded to string. We then compare the predicted and reference answer strings using the `compute_exact` and `compute_f1` functions that were provided in the official SQuAD evaluation script.

4.3 Experimental details

For the three default project tasks, we ran the experiments on the Google Cloud Platform, using a VM with one NVIDIA T4 GPU (argument `-use_gpu` included in all runs), and 13GB of interal memory. We used learning rate 1e-5, 10 epochs, batch size 8 and hidden dropout probability 0.3 in all our default project task experiments.

- For the upper baseline, "BERT paper, fine tune per task": This is the architecture described in "Section 3.2 - Fine-tuning BERT" of Devlin et al. (2018), trained with the `option=fine_tune` flag. The model had a BERT encoder instance fine-tuned separately for for each task.
- For the "Lower baseline BERT paper, with pretrain" we use a single BERT encoder pretrained (run with `option=pretrain`) and only update gradients for the the outer layers for each task.
- For "BERT paper, round robin, annealed sampling" we fine-tune a single BERT instance with outer layers for each of the three default project tasks. Along the way, the model that performs best (avg score) along those tasks is the one that is being kept/saved. Trained with `option=fine_tune,roundrobin`.
- "Same roundrobin + SQUAD" is the same setup as "BERT paper, round robin, annealed sampling", but SQuAD is also trained and evaluated along with the three default project tasks.
- For "Fine tune single BERT model tasks sequentially", in one epoch we first completely fine-tune a single BERT instance on sentiment, then continue to fine-tune on paragraph and finish with sentiment. Along the way, the model that performs best (avg score) along those tasks is the one that is being kept/saved. Trained with `option=fine_tune`.
- "Multi-task LoRA (rank=16)": For each minBERT layer's self-attention module's key, query, value and dense layer, we add pairs (one per task) of trainable matrices that, multiplied, form a low-rank matrix and are pooled with the BERT Layer pre-trained weights which are kept frozen. Trained with `option=lora, lora_r=16, lora_alpha=16`.
- For each other strategy we fine tune a model with a per-task BERT instance, and train the outer layers using the appropriate strategy as set in the arguments `strategy = 'crossattention' or 'siamesebert' or 'cosine'`

For the SQuAD results in Table 2, the "SQuAD: BERT paper, fine tune without the other tasks" training was done an NVIDIA V100 GPU with 15GB of internal memory; this is a larger GPU than the NVIDIA T4 used for the other experiments.

Default three tasks –> ================== Strategy and finetune vs pre-train	Sentiment Classification	Paraphrase Detection	Semantic Textual Similarity (STS)	Avg. Score	Avg. training time per epoch (hour:min)
Upper baseline Dev leader board 0.772: BERT paper, fine tune per task	0.517	0.887	0.853	0.752	0:42
Test leader-board 0.783: BERT paper, fine tune per task	0.534	0.889	0.849	0.757	N/A
Test leader-board 0.781: BERT paper, round robin, annealed sampling	0.530	0.888	0.851	0.756	N/A
Fine tune single BERT model, round robin, annealed sampling	0.510	0.883	0.848	0.747	0:21
Same roundrobin + SQuAD	0.501	0.888	0.834	0.741	1:01
LoRA (rank=16) pre-trained BERT model	0.470	0.863	0.784	0.706	0:24
Fine tune single BERT model tasks sequentially	0.215	0.859	0.879	0.651	0:43
Siamese BERT cosine similarity, fine tune per task	0.517	0.795	0.530	0.614	0:50
Siamese BERT vector pooling, fine tune per task	0.517	0.863	0.447	0.609	0:54
Cross Attention, per task	0.516	0.830	0.309	0.551	1:04
Lower baseline BERT paper, with pretrain	0.311	0.675	0.246	0.411	0:11.5

Table 1: Experimental results for 3 Default Project tasks.

Training strategy	SQuAD dev accuracy	SQuAD dev F1	SQuAD test accuracy	SQuAD test F1	Avg. time per epoch (hour:min)
SQuAD: BERT paper, fine tune without the other tasks	0.636	0.722	0.614	0.678	0:41 (V100 GPU)
SQuAD: Fine tune single BERT model, round robin, annealed sampling, including other three tasks	0.624	0.713	0.603	0.668	1:01

Table 2: Experimental results for SQuAD task.

4.4 Results

See Table 1 for results on the default project tasks. Green indicates an efficient approach, using a single BERT model and fast training. Other color indicates an inefficient approach.

Our "SQuAD: BERT paper, fine tune without the other tasks" dev/test accuracy and F1 score are lower than those obtained in Devlin et al. (2018) using their BERT-LARGE model, which is to be expected, since our minBERT model corresponds to BERT-BASE from Devlin et al. (2018), which has one third the number of parameters as BERT-LARGE. Devlin et al. (2018) did not evaluate their BERT-BASE model on SQuAD 2.0.

5 Analysis

The results showed that round robin with Annealed Sampling had an avg score of just 0.005 below our upper baseline on the dev leader board. A different random seed, or submitting on the test board could reverse that difference. Most notably, this round robin approach was more efficient by two metrics: training was twice as fast, and the memory footprint was only one third.

Adding QA training on SQuAD as a fourth task with round robin lowers the average score by 0.006 for the existing three tasks, while lowering the score on the QA task by 0.012 compared to single task fine tuning on SQuAD. This may be because of slightly conflicting loss gradients between SQuAD and the default project tasks, especially between SQuAD and sentiment analysis, although we did not run experiments to test this hypothesis.

Our lower baseline, only training the outer layer with the linear classifiers while keeping the weights of the single BERT model frozen, was easy to beat as far as performance is concerned. It shows that despite having task specific outer layers, much of the performance/score improvement comes from fine tuning the BERT encoder weights.

The sentence comparison strategies that deviated from the default BERT paper, Siamese BERT (cosine and vector pooling) and cross attention, were not intended as an efficient option (one of the goals of our project) but as an attempt to beat the upper baseline. As mentioned above, fine tuning the encoder weights for the tasks is very important for getting high scores. It turned out that adding more sophisticated task-specific outer layers actually had worse performance, most notably on the sentence similarity task.

With about 0.05 below the upper baseline, LoRA turned out to be a good and very efficient alternative to the upper baseline. With rank 16 we increased the size of the four of the attention weights (query, key, value and dense output) by about 4 percent (2 * 16/768).

6 Conclusion

In our experiments we found that Round Robin with Annealed Sampling has been the most efficient as well as best performing (in terms of accuracy/score) for fine-tuning a single BERT model for multiple tasks. The task-specific outer layers are small and hardly contribute to the overall memory footprint.

Adding a fourth task, QA, while fine tuning Round Robin only slightly decreased the accuracy of the existing three tasks. The accuracy of all tasks, including QA, decrease only slightly compared to task-specific (per task) fine tuning.

Another model architecture, with one BERT model encoder per task, performed slightly better on the dev leader board, but the difference is very small. Training took twice as long, and the memory footprint is three times as large.

The multi-task LoRA model performs reasonably well and is still quite efficient as far as training time and memory footprint is concerned. The big efficiency gains of LoRA come in when in the future new tasks are being added. Because the new LoRA weights and outer layer can simply be plugged in and are completely independent from the rest of the model, there is no need to simultaneously train on the other tasks [3].

Other strategies, Siamese BERT and cross attention were much less efficient. They performed reasonably well on the Paraphrase task but somehow poorly on Sentence Similarity. For cosine similarity it was probably because we had guessed the scale, rather than having an additional linear layer learn it. One reason cross attention would not do better is probably that the self attention in the default strategy is already doing a form of cross attention with the two sentences separated by [SEP]. We probably could get better results if we do the cross attention bidirectionally. [4] [5]

[3]For that reason also no need to keep the same datasets that were used to train previous versions of the model

[4]At least one benefit of using two parallel BERT encoders is that the context window is doubled (sentence twice as long) in comparison tasks.

[5]Yet another method we tried, sentiment task pre-finetuning using the Amazon Polarity data from HuggingFace did not really make a difference on just the SST sentiment analysis task, and hence was discontinued from further experimentation.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

Asa Cooper Stickland and Iain Murray. 2019. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

A APPENDIX Siamese BERT Vector Pooling and Cosine Similarity

In vector pooling we combine the two encoder outputs from the two sentences:

- u: the pooler output of sentence 1
- v: the pooler output of sentence 2
- $|u - v|$: the element-wise difference between the two pooler outputs

In the softmax classifier we then have trainable weight $W_t R^{3n} k$, and we get output probabilities $softmax(Wt(u, v, |u - v|))$.

For Cosine Similarity: We currently use six classes, from 0 to 5, to predict the sentence similarity. So far this has provided a very good result. A little better result might be obtained with linear regression, since the scale for similarity is 0.0 through 5.0. However, that will mean adapting the code for the various strategies that have been implemented.

Cosine similarity cos is translated to logits the following way:

- Paraphrase, labels [0,1]:
  [1-cos, cos]
- Sentence similarity, labels [0,1,2,3,4,5]:
  [1-cos, 1 - abs(cos+0.2), 1 - abs(cos-0.1) , 1 - abs(cos-0.4) , 1 - abs(cos-0.7), cos]

It is not clear how humans interpret sentence similarity. A cosine similarity of 0 would indicate that the sentences are not related at all, whereas a value of -1 would indicate that the two sentences are the complete opposite. Therefore, logits 2, 3, 4 and 5 favor a cosine similarity > 0.