A New Computationally Efficient Method to Tune BERT Networks – Transfer Learning

Zian Wang

School of Computing and Information, University of Pittsburgh,
Pittsburgh, Pennsylvania, USA,15213
ziw42@pitt.edu

Abstract. BERT is a pre-trained language model that achieves state-of-the-art performance on natural language processing (NLP) tasks. Once it was published, it quickly became one of the most popular models in the NLP field. The official recommended method for applying BERT to downstream tasks is fine-tuning. However, we argue that transfer learning is also a very practical approach to applying BERT, and this approach has its own advantages compared to fine-tuning. In this paper, we explore the advantages of the transfer learning approach through the method that uses transfer learning and fine-tuning approaches separately to apply BERT to the same eight GLUE benchmark tasks and compare the training time spent and model performance scores, such as accuracy or F1 score, of the two approaches. Finally, this paper finds that among all eight GLUE tasks and on small training sets, transfer learning saves 30% to 50% of the time compared to a fine-tuning approach to train the models on the data of downstream tasks and can achieve very similar performance. Besides, for the same amount of time, transfer learning can obtain higher performance scores than fine-tuning on larger training sets. In conclusion, on small training sets, transfer learning has a huge advantage in time consumption under the premise of approximate performance, and it also performs better on large training sets under the same length of time, compared to fine-tuning.

Keywords: BERT, transfer learning, fine-tuning, NLP

1. Introduction

Natural Language Processing(NLP) is a branch of machine learning that is applied to many areas, such as machine translation and sentiment analysis. BERT, which stands for Bidirectional Encoder Representations from Transformers[1], is a pre-trained model in the NLP field which was published in 2018. Once released, BERT outperformed most mainstream NLP models prior to it. Therefore, it quickly became one of the most used models for solving NLP tasks[1], and how to efficiently tune and apply it to various tasks has gradually become an important problem in the NLP field in recent years.

The common approach to applying BERT to downstream tasks is fine-tuning. In this approach, the BERT network is initialized by pre-trained parameters, and all these parameters are further trained on data of the downstream task for several epochs. In the machine learning field, transfer learning is a method that can apply a network's knowledge gained during solving a previous problem to another related problem.

Pan et al. introduced basic ideas of transfer learning and addressed the problem of what to transfer and how to transfer [2]. Zhuang et al. gave further explanations and application examples that are closer to current trends [3]. Sun et al. conducted several experiments to test the advantages and

disadvantages of different fine-tuning methods [4]. Hao et al. fine-tuned BERT on several datasets and gave a detailed visualization of loss landscapes and optimization trajectories of fine-tuning BERT [5–6].

In this paper, our basic idea is to use the transfer learning approach to apply BERT to downstream tasks. We directly apply most of BERT's pre-trained parameters to solve the downstream tasks, and only train a few layers of parameters. The trainable parameters will be 8.33% to 25% of the total parameters. Even though the fine-tuning approach has reduced the training time and hardware required to apply to downstream tasks, we argue that transfer learning is a more computationally efficient way to apply a BERT network to downstream tasks. Through transfer learning, we can save lots of time by getting similar high-level performance and get better performance in the same length of time, especially under limited hardware conditions.

2. Methodology

2.1. BERT

BERT is a pre-trained language representation model that is based on transformers. It was pre-trained on bidirectional representations from the unlabeled text by two tasks: Masked LM and Next Sentence Prediction. It was trained on two corpora with a total of 3300 million words. Therefore, we believe that in this pre-training step, BERT models have already learned a lot, which gives us the possibility to use transfer learning in applying BERT. In this paper, we focus on the BERT-base model, which has 12 layers of encoders with a hidden size of 768, 12 self-attention heads, and 110M total parameters. To apply BERT to downstream tasks, the most common approach is fine-tuning.

2.2. Fine-tuning

Input-output BERT models can be easily applied to downstream tasks by plugging in input layers and output layers of the tasks, and in a fine-tuning approach, by tuning all parameters in all layers. Since neural networks store knowledge in the parameters of neurons [7], through this workflow, the knowledge learned in the pre-training phase can be loaded and can be further tuned by specific tasks to fit them. Fine-tuning is a time-and-hardware-inexpensive approach. According to Devlin et al. (2017), the results in the paper can be reproduced in several hours on a GPU. The process of fine-tuning is shown in Figure 1.

However, if we want to train a BERT model on a huge dataset, or there is no GPU available, or we already have a BERT model that was pre-trained on a similar task compared to ours, and we want to test the model before we make further decisions or tune some hyper-parameters, fine-tuning will be a time-consuming approach. To solve this problem, we propose a new approach for applying BERT to downstream tasks: transfer learning.

2.3. Transfer learning in BERT

Transfer Learning is widely used in Machine Learning. As its name says, it can transfer learned knowledge from one domain to another related domain to improve the learner's performance [6]. In the case of applying BERT models, we want to transfer the models' knowledge learned during the pre-training phase to our downstream tasks. Without any training, we will directly use most of the parameters in BERT to solve our task.

A common workflow for transfer learning used in other problems is applied in this paper, and it is also listed in Figure 1. The first step is to initialize the BERT network with its pre-trained parameters. This step loads all the pre-trained knowledge. Then the second step is freezing most of its layers, only leaving a few layers trainable. The parameters in frozen layers will remain unchanged during training. Only the trainable layers' parameters can be trained. Therefore, most of the loaded knowledge is preserved during the training process. It is this step that makes transfer learning different from fine-tuning. The third step of the workflow is adding input and output layers. This is for adapting BERT models to different downstream tasks. For example, we will add one single unit output dense

layer for binary classification tasks [8]. Then the final step is training all trainable parameters and evaluating the performance of our models. Through this workflow, we can make use of BERT's pre-learned knowledge and apply it to our downstream tasks.

We can see that in transfer learning training, we freeze most of the parameters, which means that compared to the fine-tuning approach, there will be many fewer parameters needed to be trained. Therefore, training in transfer learning will be quicker than in a fine-tuning approach. However, since not all parameters are trained by task-specific data, the models will perform worse. We argue that in this trade-off, we can achieve a large reduction in training time with very little performance degradation.

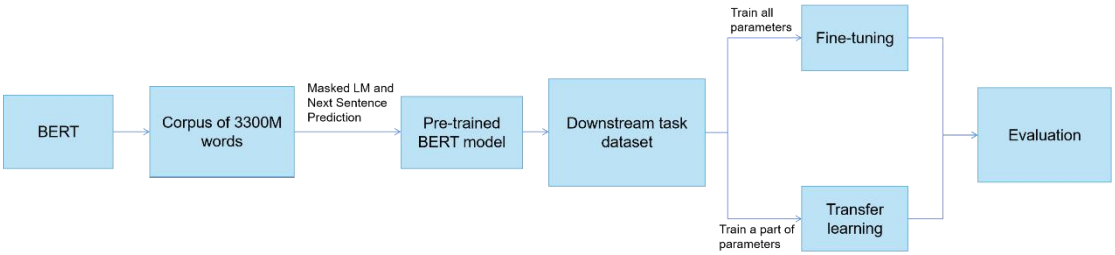

Figure 1. The process of transfer learning and fine-tuning

3. Experiments

3.1. GLUE

The General Language Understanding Evaluation (GLUE) benchmark is a set of natural language understanding (NLU) tasks[9]. It includes question answering, sentiment analysis, and textual entailment tasks to test if a model has a "general, flexible, and robust"[9] ability to understand language and execute a variety of NLU tasks in different domains, rather than being good at tasks in one specific domain.

The reason for this paper's using GLUE benchmark tasks is that in the BERT paper [10], they used the GLUE benchmark to test BERT's ability, so we follow the original paper's method to examine the difference between the transfer learning approach and the fine-tuning approach. The advantage of using the GLUE benchmark is that we can obtain the performance under several basic NLU tasks, so the conclusion can cover most of the conditions in the NLP field.

We apply BERT to eight GLUE benchmark tasks. Seven of them can be treated as classification tasks, and the last one is a regression task. We add an output layer to receive the predicted value of the model. The following are the tasks in the GLUE benchmark that we will use in this paper.

The Corpus of Linguistic Acceptability (CoLA) is a single-sentence binary classification task that is published by the NYU MLL lab. This task requires models to predict whether a sentence in English is grammatically "acceptable to native speakers"[10].

The Stanford Sentiment Treebank (SST-2) is published by the Stanford NLP group. The target of this task is to classify single sentences into two sentiment labels, so the sentiment analysis ability is tested[11].

The Microsoft Research Paraphrase Corpus (MRPC) is a sentence-pair binary classification task that aims to judge whether two sentences in the same pair have the same semantics [12].

The Semantic Textual Similarity Benchmark (STS-B) is a task that aims to score the semantic similarity of two sentences on a scale of 1 to 5. It is based on a text database collected from image captions, news headlines, and user forums [13].

Quora Question Pairs (QQP) is a task that aims to judge whether two questions have equivalent semantics. This task is based on a dataset built from the Quora website [14].

Multi-Genre Natural Language Inference (MNLI) is an entailment classification task that gives pairs of sentences. The model needs to classify the second sentence into one of three labels based on the first sentence. This dataset is a large-scale dataset that contains data from several domains. There are two versions of this dataset: MNLI-Matched and MNLI-Mismatched. The training dataset and test

dataset from the matched version are from the same source; those of the mismatched version are from a different source [15].

Question Natural Language Inference (QNLI) is from the Stanford Question Answering Dataset (SQuAD). This dataset gives models a question-sentence pair. The goal of the task is to judge whether the sentence contains the answer to the question in the same pair [16].

3.2. Setup

We use the BERT-based pre-trained model in fine-tuning and transfer learning approaches separately in this paper. For the transfer learning approach, we set four groups of trainable layers. We made only the last layer trainable and trained the models, then the last two layers trainable and trained the models, the last three layers trainable and trained, and finally the last four layers trainable and trained. For the fine-tuning approach, we simply use the method given by the BERT paper. We train models in the exact same hyper-parameter setting except for learning rate. We set the learning rate differently because for fine-tuning, the official BERT paper recommends using a small learning rate, so we set it to 2e-5, which is recommended by the BERT official team. However, for transfer learning, increasing the learning rate helps improve the performance of our models, so we set the learning rate to 1e-3 for all transfer learning experiments. To exclude the effect of the learning rate, we will also use a fine-tuning approach at the learning rate of 1e-3 and use transfer learning at the learning rate of 2e-5.

We run all the GLUE benchmark tasks and evaluate their performance on our local devices. Since we just want to make comparisons between local models, uploading them to the GLUE website for evaluation is unnecessary. We use 90% of the training data provided by GLUE to train our model and the rest, 10%, as a validation set. We test models by using development data, which is split by GLUE for users to evaluate their models locally. For each setting, we train the models until their performance score on the validation set stops increasing by a small threshold for two consecutive epochs. We test the models' performance on test data after each epoch. We will run each set of settings twice and calculate the average score as the final score in this paper.

We will use the same metrics used in the BERT paper to evaluate the performance of each task. The F1 scores are used for the QQP and MRPC tasks; the Spearman correlation coefficient for the STS-B task; and accuracy for the remaining tasks. Accuracy shows the fraction of correctly predicted instances among all the instances. Its formal definition is formula 1.

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \quad (1)$$

F1 score combines the precision and recall of the model, its formula is shown in formula 2, where TP is True Positive count, FP is False Positive count, and FN is False Negative count.

$$F_1 = \frac{TP}{\text{TP}+\frac{1}{2}(FP+FN)} \quad (2)$$

The Spearman correlation coefficient(ρ) is used in the STS-B task. Its definition used in this paper is formula 3, where di is the difference between the ranks of prediction and the ranks of true labels, and n is the total number of rows [17].

$$\rho = 1 - \frac{6\sum d_i^2}{\text{n}(n^2-1)} \quad (3)$$

All experiments are implemented in TensorFlow 1.15.0, Python version 3.8. We only use the uncased BERT-base model [13] as the pre-trained BERT model in all experiments. We use the tokenizer and embedding method provided by the BERT official code. For all the training, we set the batch size to 1, and all models are tuned on a single NVIDIA RTX 3060 GPU (16 GB RAM). The detailed hyper-parameter and trainable layer settings are shown in Table 1.

Table 1. Hyperparameters setting in experiment

Parameter	Value
Batch size	1
Max number of epochs	30
Dropout rate	0.1

Learning rate	2e-5 / 1e-3
Max sequence length	72
Optimizer	Adam

4. Results and discussion

Firstly, to exclude the influence of learning rate, we want to analyze how learning rate influences the performance score of models. For the fine-tuning approach, the BERT official team recommends using a small learning rate: 2e-5 as the default value. However, transfer learning often requires a higher learning rate to achieve better performance. Therefore, we use 1e-3 as the higher rate. We use the transfer learning and fine-tuning approaches to train models at those two learning rates separately. The results are shown in Figure 1. For fine-tuning, if we use a higher learning rate, the average score will be 64% lower than using the recommended learning rate. Besides, if we use transfer learning, the average score will be lower, at around 34%. For training time used, there is no considerable difference between these two learning rates under all conditions. Since we want to exclude the effect of the learning rate, we will only compare the better performance scores of transfer learning and fine-tuning under these two learning rates.

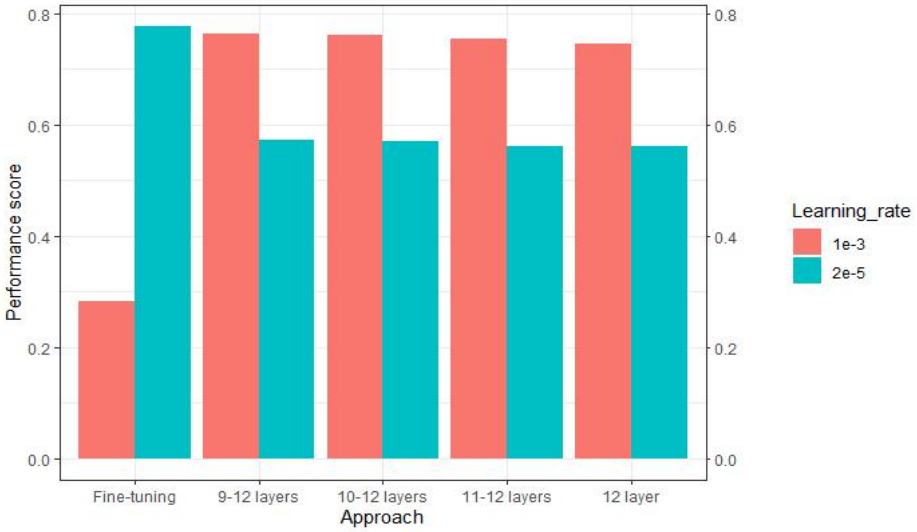

Figure 2. Comparison of average performance scores in two different learning rates. (The score on the y axis is the average value of scores of eight tasks.)

After excluding the influence of learning rate, we can start to analyze the difference between transfer learning and fine-tuning. The training time used and the performance score after each epoch are listed in table 2 to 9.

Firstly, we only consider the case where the training of both approaches reaches the epoch where the performance score on the validation set no longer increases above a tiny threshold. Transfer learning does not necessarily save much time in the first four tasks: MNLI, QQP, QNLI, and STS-2. We can see that although the transfer learning approach uses about 60% of the time to train an epoch compared to the fine-tuning approach, it often takes more epochs to reach the stopping threshold. Therefore, in the QQP task, training 9 to 12 layers and 10 to 12 layers only saves 6.43% and 3.19% of training time, respectively, compared to the fine-tuning approach to meet the stopping threshold. In QNLI and SST-2 tasks, only training 9 to 12 layers takes more time to coverage than a fine-tuning approach. However, if we only train the last two layers, it can save 37.3% of training time on average, and if we only train the last one layer, it can save 49.61% of the time on average. Transfer learning, in which training is divided into three or four layers, also saves a significant amount of time. Only training 9 to 12 layers and only training 10 to 12 layers saves 33.48% and 53.24% of training time, respectively, compared to the fine-tuning approach. Training only the last two and one layer saves

44.15% and 48.15% of the time on average, which is close to the saving time of training the last three and last four layers.

Then if we look at the performance score, we can find that also in the first four tasks, transfer learning where training 9 to 12 and 10 to 12 layers achieves a highly similar performance score to the fine-tuning approach. In these tasks, the final performance scores of training only 9 to 12 layers and training 10 to 12 layers are merely 0.2125% and 0.4925% less on average than those of the fine-tuning approach, respectively. If only trained on the last two layers or the last one layer, the deductions are tougher, which are 1.67% and 2.17% on average. In the remaining four tasks, there is a relatively larger difference between the performance scores of the two approaches. The performance score of training 9 to 12 layers and 10 to 12 layers is respectively 2.36% and 2.52% less on average than that of the fine-tuning approach. For only training the last two or one layer, the deductions are 2.79% and 4.25% on average. The reason why the comparison results between transfer learning and fine-tuning are different in the first four tasks and the last four tasks is that the first four tasks have much more training data than the last four tasks. The training data for the first four tasks ranges from 67000 to 392000 samples, while the training data for the remaining four tasks ranges from 2500 to 8500 samples.

From the above results, if one wants to train a BERT model on downstream tasks until its performance score no longer increases noticeably, we can draw two conclusions. The first one is that if the downstream task has a large training set, which has around 100,000 or even more samples, we should choose only to train the last two or one layer to get a training time saving of around 40%. However, we should also notice that there will also be a performance score deduction of around 2%. There is no apparent reason to train the last four or three layers on such large datasets, as although their deduction on performance scores is less than 0.5%, their saving on training time is not worth considering. The second one is that, in the case of small training sets that have less than 10,000 samples, it is worth using transfer learning that trains the last four or three layers because it can trade only around 2.4% of performance score deduction for saving 33% to 50% of training time. However, we must notice that although transfer learning on small training sets can save up to 50% of training time, the training time was already short before it was shortened. For example, in the experiment on the RTE task, only training 9 to 12 layers saved 1526 seconds, but the original training time taken by the fine-tuning approach is only 3263 seconds. Therefore, merely 1737 seconds were saved if people only trained the model once. However, in this case, transfer learning is still practical when people want to train a BERT model several, or even tens of times, for purposes such as tuning the hyper-parameters of the model. In this case, the transfer learning approach can save tens of hours.

Table 2. Performance scores and training time collected after each epoch of the MNLI task. (The numbers before slashes are the performance scores, the ones after are used training time.)

MNLI-matched (392k)								
Approaches	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5	Epoch 6	Epoch 7	Epoch 8
Fine-tuning	0.8133/ 38866	0.8203/ 77734	0.8236/ 116599	0.8275/ 155463	0.8285/ 194331	0.8299/ 233197	0.8307/ 272063	
9-12 Layers	0.8156/ 22783	0.8212/ 45567	0.8249/ 68348	0.8269/ 91134	0.8287/ 113914	0.8283/ 136700	0.8299/ 159481	0.83/ 182265
10-12 Layers	0.807/ 20874	0.8151/ 41750	0.819/ 62623	0.8205/ 83496	0.821/ 104371	0.8205/ 125244	0.8238/ 146118	0.8247/ 166992
11-12 Layers	0.7944/ 18437	0.8029/ 36875	0.8065/ 55312	0.8092/ 73747	0.8121/ 92187	0.8141/ 110624	0.8148/ 129058	

12 Layer	0.7837/ 16925	0.7896/ 33854	0.7911/ 50783	0.7928/ 67708	0.7949/ 84637	0.7956/ 101562	0.7965/ 118490	

Table 3. Performance scores and training time collected after each epoch of the QQP task. (The numbers before slashes are the performance scores, the ones after are used training time.)

QQP (363k)								
Approaches	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5	Epoch 6	Epoch 7	Epoch 8
Fine-tuning	0.7307/ 40040	0.7328/ 80080	0.7364/ 120120	0.7381/ 160160				
9-12 Layers	0.7251/ 21496	0.7347/ 42993	0.7341/ 64490	0.7351/ 85985	0.7364/ 107481	0.7371/ 128975	0.7372/ 150473	
10-12 Layers	0.7229/ 19402	0.7294/ 38806	0.7319/ 58205	0.7345/ 77607	0.7354/ 97009	0.7354/ 116412	0.7363/ 135815	0.7365/ 155215
11-12 Layers	0.723/ 17506	0.7238/ 35017	0.7299/ 52526	0.7326/ 70034	0.7323/ 87541	0.7339/ 105049		
12 Layer	0.7167/ 15735	0.718/ 31472	0.7229/ 47205	0.7252/ 62939	0.7262/ 78675	0.727/ 94412		

Table 4. Performance scores and training time collected after each epoch of the QNLI task. (The numbers before slashes are the performance scores, the ones after are used training time.)

QNLI (108 k)								
Approaches	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5	Epoch 6	Epoch 7	Epoch 8
Fine-tuning	0.8645/ 10922	0.87/ 21844	0.8742/ 32767	0.8748/ 43688	0.8757/ 54609	0.8754/ 65534	0.875/ 76453	
9-12 Layers	0.858/ 6132	0.8674/ 12267	0.8709/ 18403	0.8724/ 24537	0.8739/ 30670	0.8739/ 36806	0.8744/ 42940	0.8714/ 49072
10-12 Layers	0.8535/ 5527	0.8638/ 11059	0.8681/ 16586	0.8699/ 22117	0.8695/ 27647	0.8672/ 33174	0.868/ 38704	0.8672/ 44234
11-12 Layers	0.8393/ 4978	0.8435/ 9962	0.8442/ 14942	0.8465/ 19920	0.8429/ 24900	0.842/ 29881	0.8425/ 34859	0.8433/ 39840
12 Layer	0.8349/ 4317	0.8401/ 8637	0.8406/ 12956	0.8439/ 17272	0.8429/ 21591	0.8469/ 25909	0.8473/ 30227	0.8473/ 34543

Approaches	Epoch 9	Epoch 10	Epoch 11	Epoch 12	Epoch 13	Epoch 14	Epoch 15
Fine-tuning							

9-12 Layers	0.8718/ 55206	0.8697/ 61340	0.8697/ 67475	0.8718/ 73609	0.8706/ 79743	
10-12 Layers	0.867/ 49760	0.8672/ 55291	0.8655/ 60820	0.8659/ 66347	0.864/ 71876	0.8664/ 77407
11-12 Layers	0.8418/ 44822	0.841/ 49800				
12 Layer						

Table 5. Performance scores and training time collected after each epoch of the SST-2 task. (The numbers before slashes are the performance scores, the ones after are used training time.)

SST-2 (67k)								
Approaches	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5	Epoch 6	Epoch 7	Epoch 8
Fine-tuning	0.915/ 6872	0.9196/ 13749	0.9206/ 20621	0.9228/ 27496	0.9194/ 34370	0.9206/ 41244	0.9194/ 48119	
9-12 Layers	0.9101/ 3924	0.916/ 7852	0.9198/ 11777	0.9197/ 15702	0.9203/ 19625	0.9189/ 23551	0.9191/ 27477	0.9204/ 31399
10-12 Layers	0.9058/ 3678	0.9102/ 7356	0.9136/ 11033	0.9144/ 14713	0.9159/ 18389	0.9171/ 22070	0.9148/ 25746	0.9171/ 29423
11-12 Layers	0.9022/ 3176	0.9011/ 6356	0.9033/ 9532	0.9056/ 12709	0.9033/ 15884	0.9068/ 19061	0.9079/ 22241	0.9068/ 25415
12 Layer	0.9056/ 2882	0.8999/ 5765	0.9045/ 8646	0.9056/ 11528	0.9033/ 14410	0.9033/ 17293	0.9033/ 20176	0.9033/ 23056
Approaches	Epoch 9	Epoch 10	Epoch 11	Epoch 12	Epoch 13	Epoch 14	Epoch 15	
Fine-tuning								
9-12 Layers	0.9208/ 35326	0.9192/ 39252	0.9195/ 43177	0.9172/ 47099	0.9169/ 51025			
10-12 Layers	0.9148/ 33102	0.9171/ 36781	0.9159/ 40459					
11-12 Layers	0.9068/ 28593	0.9056/ 31769	0.9068/ 34948					
12 Layer	0.9056/ 25940							

Table 6. Performance scores and training time collected after each epoch of the CoLA task. (The numbers before slashes are the performance scores, the ones after are used training time.)

CoLA(8.5k)				
Approaches	Epoch 1	Epoch 2	Epoch 3	Epoch 4
Fine-tuning	0.5018/917	0.5089/1839	0.5103/2757	0.5106/3676
9-12 Layers	0.5003/501	0.5017/1003	0.5029/1502	0.5028/2006
10-12 Layers	0.4982/465	0.4997/931	0.501/1397	0.5013/1866
11-12 Layers	0.4981/435	0.4985/872	0.4989/1307	0.4991/1745
12 Layer	0.4977/387	0.4979/778	0.4982/1167	0.4979/1558

Table 7. Performance scores and training time collected after each epoch of the STS-B task. (The numbers before slashes are the performance scores, the ones after are used training time.)

STS-B (5.7k)								
Approaches	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5	Epoch 6	Epoch 7	Epoch 8
Fine-tuning	0.8307/ 549	0.837/ 1102	0.84/ 1653	0.8401/ 2203	0.8405/ 2755	0.8403/ 3306	0.8401/ 3857	0.8394/ 4408
9-12 Layers	0.8034/ 348	0.8157/ 696	0.8186/ 1045	0.8222/ 1393	0.8211/ 1741	0.8207/ 2090	0.8234/ 2435	0.8157/ 2785
10-12 Layers	0.8143/ 306	0.8154/ 616	0.8171/ 923	0.8162/ 1231	0.8176/ 1540	0.8166/ 1848		
11-12 Layers	0.8046/ 268	0.8125/ 540	0.815/ 809	0.8113/ 1076	0.8087/ 1344	0.8052/ 1613	0.8007/ 1884	0.7984/ 2151
12 Layer	0.7947/ 251	0.798/ 503	0.8026/ 755	0.797/ 1007	0.79812/ 1261	0.797/ 1512	0.7929/ 1766	0.793/ 2017

Approaches	Epoch 9	Epoch 10	Epoch 11	Epoch 12	Epoch 13	Epoch 14	Epoch 15
Fine-tuning	0.8407/ 4961						
9-12 Layers	0.822/ 3132	0.8205/ 3482	0.8222/ 3830	0.8248/ 4176			
10-12 Layers							

11-12 Layers	0.8017/ 2423	0.8016/ 2691	0.8025/ 2960	0.8047/ 3228	0.8088/ 3499	0.8068/ 3766	0.8075/ 4034
12 Layer	0.7943/ 2267	0.7951/ 2519	0.7945/ 2774	0.7947/ 3024	0.794/ 3278		

Table 8. Performance scores and training time collected after each epoch of the MRPC task. (The numbers before slashes are the performance scores, the ones after are used training time.)

MRPC (3.5k)

Approaches	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5	Epoch 6	Epoch 7	Epoch 8
Fine-tuning	0.8765/ 343	0.8884/ 689	0.8875/ 1037	0.8784/ 1382	0.8915/ 1725	0.89/ 2071	0.8864/ 2416	0.8837/ 2762
9-12 Layers	0.8529/ 216	0.865/ 433	0.8753/ 651	0.8656/ 868	0.8694/ 1086	0.8665/ 1303	0.8676/ 1521	0.8699/ 1737
10-12 Layers	0.8474/ 194	0.8638/ 394	0.8621/ 589	0.8534/ 783	0.86/ 982	0.8622/ 1178	0.8618/ 1374	0.8616/ 1569
11-12 Layers	0.8512/ 172	0.8535/ 347	0.8532/ 523	0.8427/ 697	0.8559/ 870	0.8445/ 1044	0.8473/ 1218	0.847/ 1392
12 Layer	0.8481/ 161	0.8471/ 324	0.8533/ 485	0.8426/ 644	0.8543/ 804	0.851/ 968	0.8508/ 1127	0.8505/ 1289

Approaches	Epoch 9	Epoch 10	Epoch 11	Epoch 12	Epoch 13	Epoch 14	Epoch 15
Fine-tuning	0.8885/ 3106	0.8852/ 3450	0.8871/ 3796				
9-12 Layers	0.8665/ 1953	0.8654/ 2169	0.8691/ 2389	0.8713/ 2604	0.867/ 2820		
10-12 Layers	0.8572/ 1766	0.8613/ 1961	0.8597/ 2156	0.8647/ 2351	0.8638/ 2547	0.862/ 2745	
11-12 Layers	0.8486/ 1567	0.8474/ 1740	0.8492/ 1913				
12 Layer	0.8477/ 1451	0.8479/ 1611	0.8486/ 1770	0.8506/ 1934	0.8487/ 2094		

Table 9. Performance scores and training time collected after each epoch of the RTE task. (The numbers before slashes are the performance scores, the ones after are used training time.)

RTE (2.5k)

Approaches	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5	Epoch 6	Epoch 7	Epoch 8

Fine-tuning	0.5957/ 250	0.5812/ 501	0.5993/ 755	0.6101/ 1005	0.6101/ 1254	0.6065/ 1506	0.6065/ 1758	0.5993/ 2009
9-12 Layers	0.5271/ 157	0.5812/ 318	0.5886/ 474	0.5848/ 631	0.5632/ 792	0.556/ 950	0.5812/ 1108	0.574/ 1266
10-12 Layers	0.5731/ 144	0.5742/ 289	0.5771/ 435	0.5751/ 580	0.5768/ 724	0.5751/ 872		
11-12 Layers	0.5487/ 131	0.5848/ 264	0.5921/ 394	0.5704/ 526	0.5668/ 656	0.5957/ 786	0.574/ 918	0.5957/ 1048
12 Layer	0.5812/ 117	0.5415/ 240	0.5307/ 359	0.5596/ 475	0.5271/ 594	0.5379/ 716	0.5235/ 834	0.5343/ 954

Approaches	Epoch 9	Epoch 10	Epoch 11	Epoch 12	Epoch 13	Epoch 14	Epoch 15
Fine-tuning	0.5957/ 2259	0.5884/ 2510	0.6065/ 2761	0.5993/ 3014	0.6173/ 3263		
9-12 Layers	0.5487/ 1421	0.574/ 1579	0.5668/ 1737				
10-12 Layers							
11-12 Layers	0.5884/ 1179	0.5812/ 1309	0.5884/ 1442				
12 Layer	0.556/ 1072	0.5596/ 1190	0.5596/ 1310	0.5451/ 1428			

Sometimes, people only have a limited amount of time to train a model. In this case, one cannot let a model train run until its performance does not increase. Therefore, we want to analyze, under these circumstances, whether transfer learning is a practical approach to getting better performance in the same amount of training time. Figures 3–6 depict the times and results of two approaches in each of the eight tasks. We can see in the last four tasks, which have small training sets, that fine-tuning always has a better performance score than all four settings of transfer learning under the same amount of time. Besides, in SST-2 and QNLI tasks where training sets have 67000 and 108000 samples, training the last 4 layers has an extremely limited advantage in performance scores only for a short period in the beginning, and the performance scores of the other three layer settings are even lower. Therefore, if one only has a limited amount of time and wants to improve BERT's performance on training sets of this size, transfer learning is not a good idea. Then, we can look at the first two tasks that have many more training samples. In the MNLI task, only training the last four layers always performs better than a fine-tuning approach under the same length of time. We pick three time points: 50000, 100000, and 150000 seconds in the MNLI task. Among these three points, the performance scores of only training the last four layers are 0.66%, 0.39%, and 0.19% higher than those for fine-tuning.

In the QQP task, we pick 40000, 80000, and 120000 seconds, and only training the last 4 layers performs 0.29%, 0.2%, and 0.04% higher than fine-tuning at these three time points separately. We can see that as training time grows, transfer learning has less and less advantage in performance scores. Therefore, transfer learning can improve performance, and it is better to use transfer learning with less

training time. Nevertheless, the advantage of transfer learning in performance scores is small: about 0.4% and 0.3% in two tasks, so this is not a strong reason that people should use transfer learning in this case. However, since the improvement is greater as the training size gets larger, as shown also by the study of Soekhoe et al. (2016) [18], we believe the advantage of transfer learning will become apparent when applying BERT to larger datasets, such as a dataset with millions of samples. Besides, a tougher reason for one to want to use transfer learning in this situation is that transfer learning can stop more flexibly since it takes less time for one epoch to finish training. We can take the experiment of the MNLI task as an example. If people use the fine-tuning approach under these circumstances and only have 20 hours to train, they can only train for one epoch, which will stop at around 11 hours and waste the rest of the 9 hours since it is hard to stop in the middle of training for one epoch. However, if people use transfer learning that only trains the last four layers instead, they can train for three epochs, which uses about 19 hours and performs 1.16% higher on the performance score. In this case, transfer learning can make use of time and bring a considerable improvement in performance score.

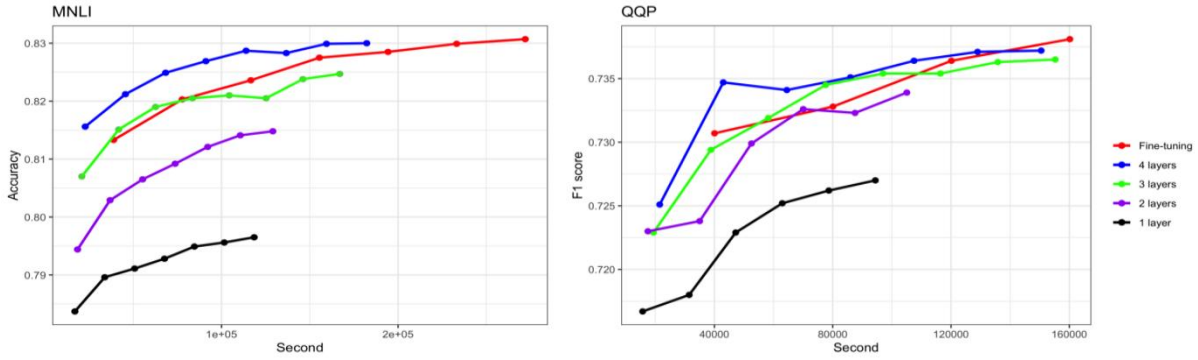

Figure 3. Training time and accuracy of transfer learning and fine-tuning on the MNLI and QQP tasks

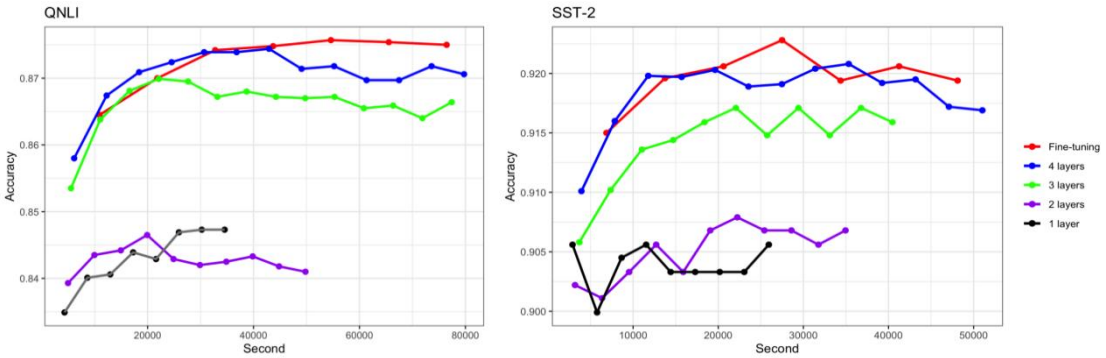

Figure 4. Training time and accuracy of transfer learning and fine-tuning on the QNLI and SST-2 tasks

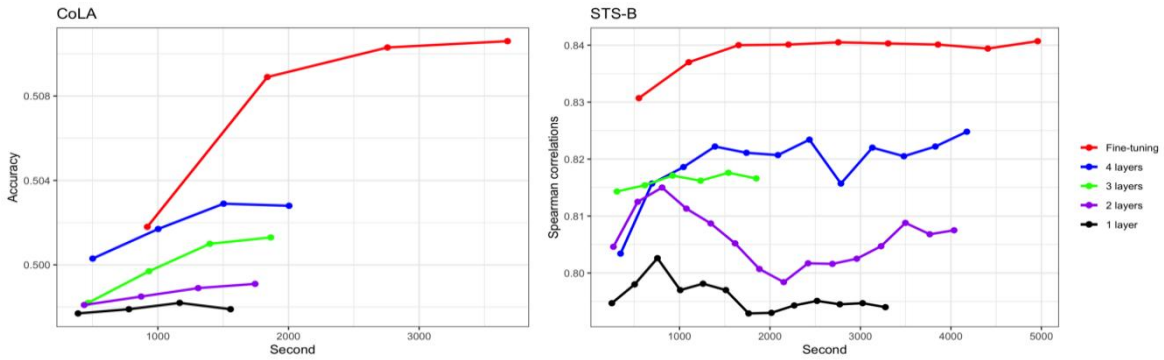

Figure 5. Training time and accuracy of transfer learning and fine-tuning on the CoLA and STS-B tasks

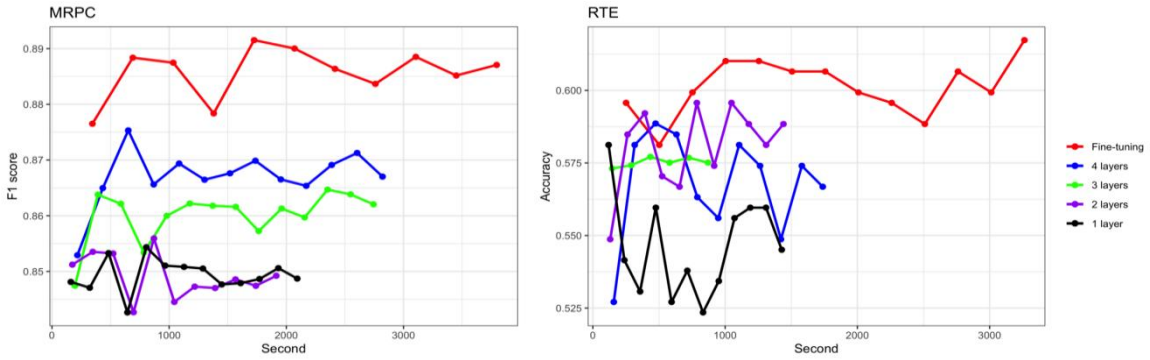

Figure 6. Training time and accuracy of transfer learning and fine-tuning on the MRPC and RTE tasks

We analyzed the advantages of transfer learning in two situations. The first is when people apply BERT to downstream tasks and want to train the model until it converges. Although for tasks that have a large training set, the advantage of transfer learning is not obvious, for tasks that have 10000 or fewer training samples, transfer learning can save from 30% to around 50% of training time. This is very useful, especially when people need to train the models many times. The second situation is when one only has a limited amount of time to apply BERT models but wants to get the best possible performance score. In this case, the advantage is greater for tasks that have a larger dataset. In the task that has the biggest training set of the GLUE benchmark: MNLI, transfer learning can give people a 1.16% performance score improvement compared to fine-tuning in the same amount of time since fine-tuning wastes 9 hours out of 20. Additionally, we believe if the training set is larger, the advantage will become greater.

5. Conclusion

In this paper, we apply the BERT-base model to 8 GLUE benchmark tasks by using both the transfer learning approach and the most common approach, fine-tuning. We exclude the influence of learning rates by training models on two learning rates and find the proper rates for two approaches to only compare the better performance scores of both. By analyzing the results, we find transfer learning can save up to 50% of training time when applying BERT to small training sets and can also obtain better model performance in the same length of time if the training set is large. Therefore, in situations where people need to train BERT models many times or only have a small amount of time, transfer learning can be used to save time or achieve better performance to reduce the burden of insufficient hardware conditions.

In this paper, we have two limitations. The first is that we only run each training two times due to hardware and time considerations. This might make the average performance score we got from tasks that have fewer than 10,000 training samples a little serendipitous. However, since the performance scores fluctuate within 1%, this limitation does not influence our conclusions. The second limitation is that we only train the last four layers for transfer learning. We believe making more layers trainable will make the results of transfer learning move closer to fine-tuning. In the future, research about the influence of trainable layers' position is needed since we only place all the trainable layers at the end of the BERT network. Besides, experiments on a larger dataset that has millions of samples are also important to test how transfer learning can help training on huge datasets on personal computers.

References

[1] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[2] Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. IEEE Transactions on knowledge

and data engineering, 22(10), 1345-1359.
[3] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., ... & He, Q. (2020). A comprehensive survey on transfer learning. Proceedings of the IEEE, 109(1), 43-76.
[4] Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019, October). How to fine-tune bert for text classification?. In China national conference on Chinese computational linguistics (pp. 194-206). Springer, Cham.
[5] Hao, Y., Dong, L., Wei, F., & Xu, K. (2019). Visualizing and understanding the effectiveness of BERT. arXiv preprint arXiv:1908.05620.
[6] Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. Journal of Big data, 3(1), 1-40.
[7] Nirkhi, S. (2010, February). Potential use of artificial neural network in data mining. In 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE) (Vol. 2, pp. 339-343). IEEE.
[8] Shi, P., & Lin, J. (2019). Simple bert models for relation extraction and semantic role labeling. arXiv preprint arXiv:1904.05255.
[9] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461.
[10] Warstadt, A., Singh, A., & Bowman, S. R. (2019). Neural network acceptability judgments. Transactions of the Association for Computational Linguistics, 7, 625-641.
[11] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013, October). Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the 2013 conference on empirical methods in natural language processing (pp. 1631-1642).
[12] Dolan, B., & Brockett, C. (2005, January). Automatically constructing a corpus of sentential paraphrases. In Third International Workshop on Paraphrasing (IWP2005).
[13] Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., & Specia, L. (2017). Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. arXiv preprint arXiv:1708.00055.
[14] Chen, Z., Zhang, H., Zhang, X., & Zhao, L. (2018). Quora question pairs.
[15] Williams, A., Nangia, N., & Bowman, S. R. (2017). A broad-coverage challenge corpus for sentence understanding through inference. arXiv preprint arXiv:1704.05426.
[16] Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250.
[17] De Winter, J. C., Gosling, S. D., & Potter, J. (2016). Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. Psychological methods, 21(3), 273.
[18] Soekhoe, D., Putten, P. V. D., & Plaat, A. (2016, October). On the impact of data set size in transfer learning using deep neural networks. In International symposium on intelligent data analysis (pp. 50-60). Springer, Cham.