Analysis of Fine-Tuning Techniques for Text Classification Tasks

Hongyi Chen, Yufei Gao
{ hc2885, yg2133 } @ nyu.edu

I. Project description

Nowadays, deep learning models grow larger and deeper at a rapid rate, the consequence is that more time and resources are needed to train them. As a solution to this problem, a common technique is to use pre-trained models. Pre-trained models have well-built architecture, optimized parameters, also they were trained with large amounts of data. On top of that, we can finetune the pre-trained model to enhance it. Finetuning takes advantage of the base model and saves time and resources. In this project, we planned to use pre-trained BERT (Bidirectional Encoder Representations from Transformers) models. The BERT model is pre-trained by two unsupervised tasks to get bidirectional representations in unlabeled text. Pre-trained BERT models can be fine-tuned by adding only one output layer to create state-of-the-art models, without the need for many task-specific architecture modifications. And it obtained new records on eleven natural language processing tasks at the time it was published [1].

For BERT achieved excellent performance in many tasks, we hope to explore its full potential through fine-tuning and utilize our model in real-life applications. We plan to analyze different techniques of fine-tuning the BERT model for text classification tasks, and based on this, build a Deep Learning system that provides text classification services. With this service, we can easily do work such as text labeling, which can meet the needs of machine learning studies.

In this project, we first investigated appropriate hyperparameters and methods in fine-tuning BERT. By utilizing further pre-training techniques, we gained even better outcomes in different datasets. We then trained a generalized sentiment analysis model, which reached 97.5% accuracy on the Yelp Review dataset. Finally, we used the model with the best performance to empower a simple text classification RESTful API service that can predict sentiment from text.

II. Technical challenges and Solving Approaches

Fine-tuning BERT involves lots of hyperparameters, including batch size, learning rate, warmup proportion, and others. The two main hyperparameters we focused on are batch size and learning rate. We aim to optimize the hyperparameters and utilize the best model in our API service. However, inappropriate hyperparameters can cause inefficient learning or overfitting. For example, a learning rate that is too large may lead to Catastrophic Forgetting, while a learning rate that is too small may make fine-tuning less efficient. Finding the best combination of hyperparameters can be time-consuming and challenging.

Another challenge we faced is to further pre-train BERT model. The BERT model is trained in the general domain, which has a different data distribution from a specific target domain [2]. Therefore, we can further pre-train BERT with target domain data. However, it is still questionable how much further pre-train is moderate and whether further pre-training with different datasets in the same domain would have positive influences.

To solve these challenges, we first conducted experiments on fine-tuning the BERT model with different hyperparameters such as batch size and learning rate. Then, we performed another set of experiments on further pre-training with different numbers of steps. Noted that, we repeated all these experiments on different datasets to get more general results. Once we found the best hyperparameters, we further pre-trained the models with one dataset and tested on the others. The details are described in the next section.

III. Implementation Details

The first step of the experiment is preparing the dataset. We choose datasets for different subtasks in the domain of text classification. Table 1 shows the type, the number of classes as well as the number of train and test cases for each dataset. After collecting the dataset, we did some preprocessing. To be more specific, all the text was tokenized and encoded.

Type	Name	Number of Classes	Number of Train cases	Number of Test cases
Topic	AG_NEWS	4	120000	7600
Topic	DBpedia	14	560000	70000
Question	YahooAnswers	10	1400000	60000
Sentiment	YelpReviewPolarity	2	560000	38000
Sentiment	AmazonReviewFull	5	300000	65000
Sentiment	IMDB	2	25000	25000

Table 1: Basic information of datasets used

The very next step is to load data for the model. As the length of sentences in datasets varies and the BERT model only allows a maximum sequence length of 512, we truncated long sentences and padded short sentences to the maximum sequence length. Also, we generated attention masks to help the model ignore the paddings.

Then it comes to the design of our model. Since we are focused on text classification tasks, we will add a classification layer on the top of the original BERT model. According to the design in BERT, the final hidden state corresponding to the first token is used as the aggregate sequence representation for classification tasks [1]. So, we take that as the input for the classification layer. And the number of outputs is variable for different datasets, which needs to be specified when creating the model. Figure 1 gives a visualization of the structure of the model.

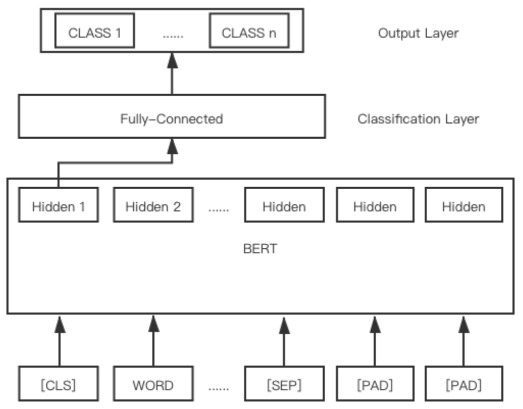

Figure 1: Structure of BERT for classification model

We divided the whole experiment into three phases, the first phase is to find appropriate hyperparameters for fine-tuning, the second one is to decide the number of steps for further pre-training, and the last is to look into the impact of different datasets.

For the first phase, we loaded and froze original pre-trained BERT model parameters, used AdamW optimizer and cosine learning rate scheduler with warm-up techniques. For each group of tests, we set different values for one of the hyperparameters and keep others the same, so we can compare the final test accuracy and find the best one among all values.

In the second phase, we first loaded original pre-trained BERT model parameters, and further pre-trained the model by different steps before saving the state dictionary. Then we only loaded the BERT layers from saved models and froze the parameters. In this way, the classification layer is independently initialized and doesn't have any relation to the further pre-trained model. It can have a different number of outputs compared to the previous one. We finetuned the model on the same dataset it was further pre-trained on to see if further pre-training is helpful and how many steps are needed.

After obtaining the appropriate hyperparameters for fine-tuning and the number of steps for further pre-training. We turned to research whether further pre-train on one dataset is beneficial to fine-tune on other datasets. We used one further pre-trained model and fine-tuned it on a series of datasets, comparing the improvements. In addition, we tried to further pre-train on more datasets to get a generalized model that works well not only on one dataset.

At this point, we got our best model on the sentiment analysis task, and we were set to build a RESTful service, making the best use of our model. We built a simple RESTful server based on the Flask project, including an API that takes a string of text and returns a predicted sentiment. There is also a sample WEB application for a better demonstration of the project.

IV. Experimental results and Observations

With the limit of computing resources, we have to use DistilBERT [3] as an alternative to BERT. DistilBERT is a distilled version of BERT created by Knowledge distillation [4] techniques, which reduce the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster.

1. Phase 1

	$1e^{-2}$	$5e^{-3}$	$2e^{-3}$	$1e^{-3}$	$5e^{-4}$
AG_NEWS	0.897631579	0.901842105	**0.904342105**	0.898684211	0.897894737
YahooAnswers	0.704533333	0.70265	**0.704083333**	0.70365	0.70335
IMDB	0.8578	**0.85804**	0.8578	0.85456	0.8438

Table 2: Best test accuracy reached with different learning rate

	16	24	32	48	64
AG_NEWS	0.899605263	0.898157895	**0.904342105**	0.897368421	0.85524
YahooAnswers	0.70165	0.702133333	**0.704083333**	0.702666667	0.701733333
IMDB	0.8554	0.85768	**0.8578**	0.855	0.85524

Table 3: Best test accuracy reached with different batch size

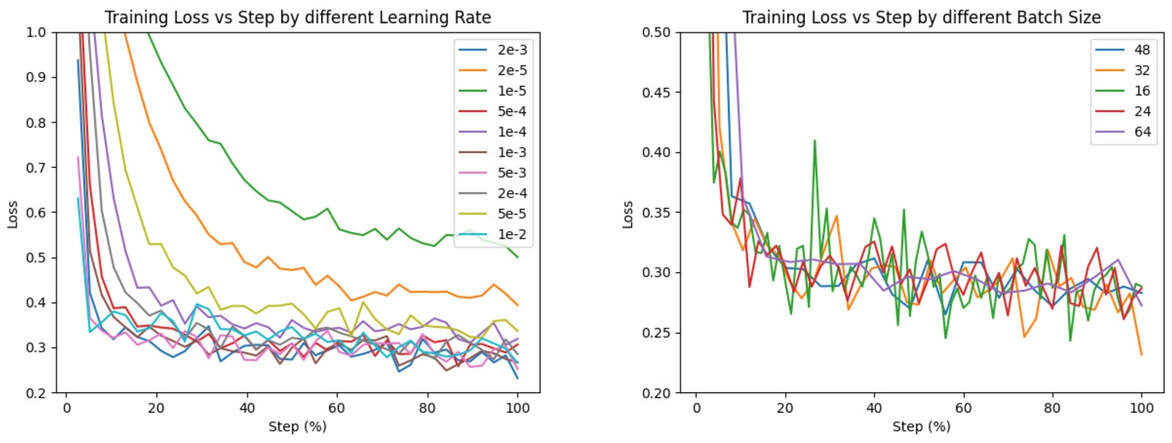

Figure 2: Training loss vs Step with different learning rate / batch size

From the test accuracy and training loss, we can identify that learning rate $2e^{-3}$ and batch size 32 works well with current settings on different types of datasets.

2. Phase 2

	0	5000	10000	20000
YelpReviewPolarity	0.90694737	0.95660526	0.97392105	**0.97507895**
AmazonReviewFull	0.505830769	0.626369231	0.635492308	**0.646123077**
IMDB	0.8578	0.93264	**0.93464**	0.93404

Table 4: Best test accuracy reached with different further pre-train steps

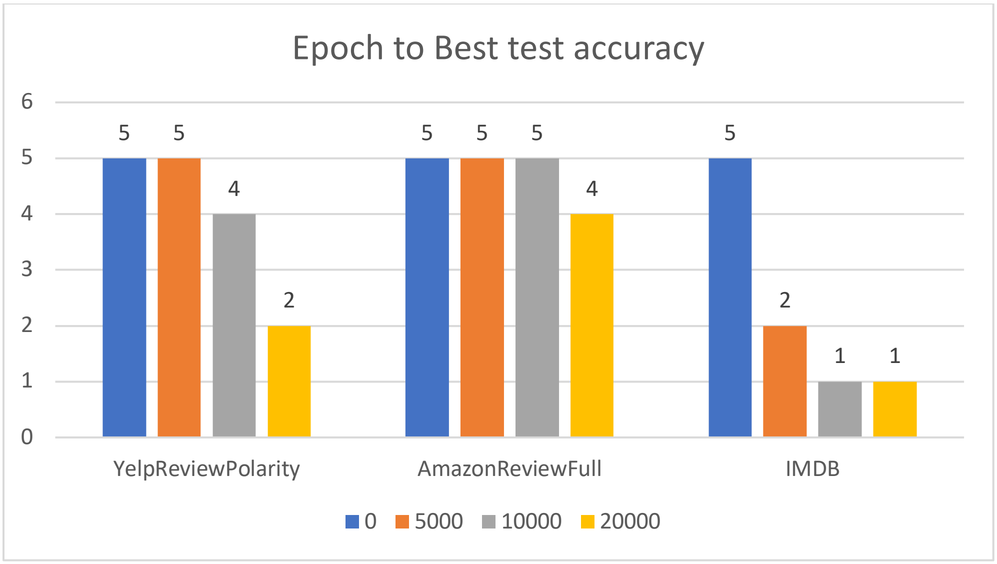

Figure 3: Number of epochs needed in fine-tune to reach best test accuracy with different further pre-train steps

From the test accuracy, we noticed that further pre-train is beneficial for model performance, and more pre-train steps do not necessarily lead to better performance especially for small datasets. Besides, further pre-train can reduce fine-tune epochs to the best accuracy.

3. Phase 3

Dataset	YelpReviewPolarity	AmazonReviewFull	IMDB
None	0.906289474	0.507292308	0.8578
YelpReviewPolarity	0.973921053	0.583553846	0.92456
AmazonReviewFull	0.957789474	**0.637030769**	0.9404
IMDB	0.930631579	0.541938462	0.93404
All except Yelp	0.950105263	-	-
All except Amazon	-	0.583692308	-
All except IMDB	-	-	0.937
All	**0.975394737**	0.625107692	**0.94276**

Table 5: Best test accuracy reached with different further pre-train datasets

By further pre-training models in different datasets, we discovered that different datasets in the same task help create a more generalized model that has decent performance on more than one dataset.

V. Conclusions

There are two main findings of fine-tuning we discovered through this project. The first is that choosing the appropriate hyperparameters is crucial in fine-tuning. The second is that further pre-training is a necessary step to improve the performance of the model. With the above findings, we achieved excellent performances on selected datasets and launched a text classification RESTful API service.

VI. Bibliography

[1] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[2] Sun, Chi, et al. "How to fine-tune BERT for text classification?." China National Conference on Chinese Computational Linguistics. Springer, Cham, 2019.

[3] Sanh, Victor, et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." arXiv preprint arXiv:1910.01108 (2019).

[4] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).

[5] Liu, Yang. "Fine-tune BERT for extractive summarization." arXiv preprint arXiv:1903.10318 (2019).

[6] Zheng, Jianming, et al. "Pre-train, Interact, Fine-tune: a novel interaction representation for text classification." Information Processing & Management 57.6 (2020): 102215.

[7] Wang, Sinong, Madian Khabsa, and Hao Ma. "To Pretrain or Not to Pretrain: Examining the Benefits of Pretraining on Resource Rich Tasks." arXiv preprint arXiv:2006.08671 (2020).

[8] He, Tianxing, et al. "Mix-review: Alleviate Forgetting in the Pretrain-Finetune Framework for Neural Language Generation Models." (2019).