NLP-DL Assignment 2 Report

Xiangyu Wang
Yuanpei College
Peking University
2300017816@stu.pku.edu.cn

1 Comparisons and analyses of different BERT-based models

I implement a complete pipeline of fine-tuning Pre-trained Language Model (PLM) following the instructions of the assignment requests. In this section, I will present the experimental results among different BERT-based models, all of which were fine-tuned during each experiment with no parameters frozen. In Sec. 1.1, I will briefly describe the configurations I utilize for different models and experiments. In Sec. 1.2, I will demonstrate the detailed experimental results including both the quantitative results and the learning curves. Ultimately, I will engage in discourse concerning comparing the performances among the experiments as well as conducting some preliminary analyses.

1.1 Configurations

I employ three different BERT-based PLMs: BERT-base(Devlin et al. [2]), RoBERTa-base(Liu et al. [5]) and SciBERT(Beltagy et al. [1]). Each model is fine-tuned on three different datasets: ACL-ARC dataset (ACL), AG's News dataset (Agnews), and SemEval-2014 Restaurant dataset (Restaurant). For each experiment, I set *batch size per device* as 32, *learning rate* as 5×10^{-5}, and *epoch* as 10.

1.2 Results

For each model with each kind of dataset, I repeatedly run five experiments that only differ in the selection of seed (I use seeds 1, 2, 3, 4, and 5, respectively), and illustrate the mean and standard deviation of these five repetitive experiments in this report. The learning curve contains the loss during training and several metrics (Accuracy, Micro-F1 and Macro-F1) of inference after each epoch of training. It's notable that for a classification problem, in which every entry is assigned with one particular label, Micro-F1 score is identical to Accuracy score. Hence, in the later sections, I will only provide Accuracy score and Macro-F1 score for the sake of brevity.

1.2.1 Quantitative results

Here I list the quantitative results of the aforementioned three models over the test sets of various datasets.
Accuracy see Tab. 1. I bold the best result of each dataset in Tab. 1. Similar bolding operations are conducted in Tab. 2 to Tab. 4.

	ACL	Agnews	Restaurant
BERT	$77.41^{\pm 2.81}$	$92.55^{\pm 0.33}$	$83.45^{\pm 0.64}$
RoBERTa	$77.70^{\pm 1.35}$	$\mathbf{93.26^{\pm 0.90}}$	$\mathbf{86.02^{\pm 0.60}}$
SciBERT	$\mathbf{81.44^{\pm 1.79}}$	$91.32^{\pm 0.25}$	$83.14^{\pm 0.50}$

Table 1: The accuracy score (%) of different models over various datasets.

Macro-F1 see Tab. 2.

	ACL	Agnews	Restaurant
BERT	$65.46^{\pm 3.15}$	$92.51^{\pm 0.33}$	$74.45^{\pm 1.15}$
RoBERTa	$66.70^{\pm 2.89}$	$\mathbf{93.21^{\pm 0.91}}$	$\mathbf{78.05^{\pm 1.24}}$
SciBERT	$\mathbf{74.82^{\pm 5.85}}$	$91.27^{\pm 0.24}$	$74.29^{\pm 1.00}$

Table 2: The macro-f1 score (%) of different models over various datasets.

1.2.2 Learning curves

I exhibit the learning curves of the aforementioned three models over various datasets during training process.
BERT The learning curves over ACL dataset: See Fig. 1

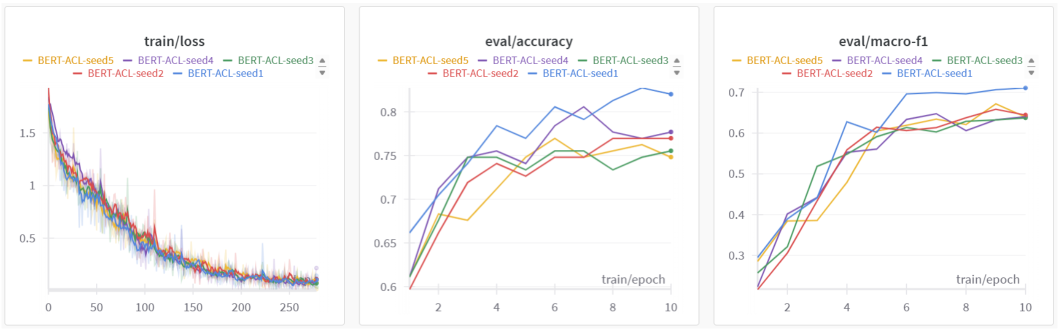

Figure 1: Learing curves of BERT model over ACL dataset.

The learning curves over Agnews dataset: See Fig. 2

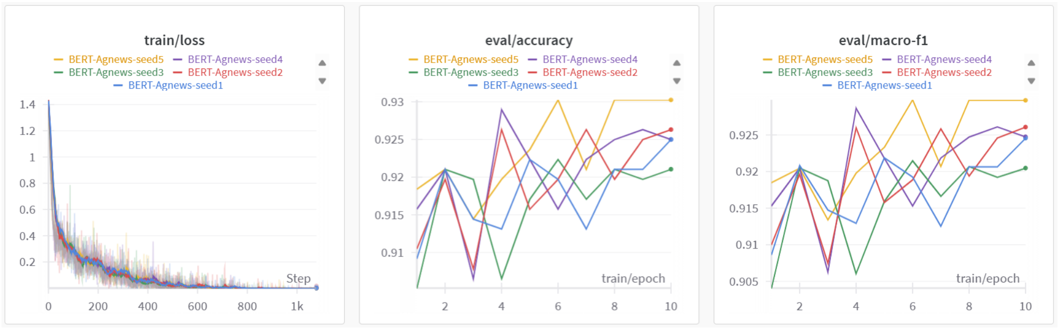

Figure 2: Learing curves of BERT model over Agnews dataset.

The learing curves over Restaurant dataset: See Fig. 3

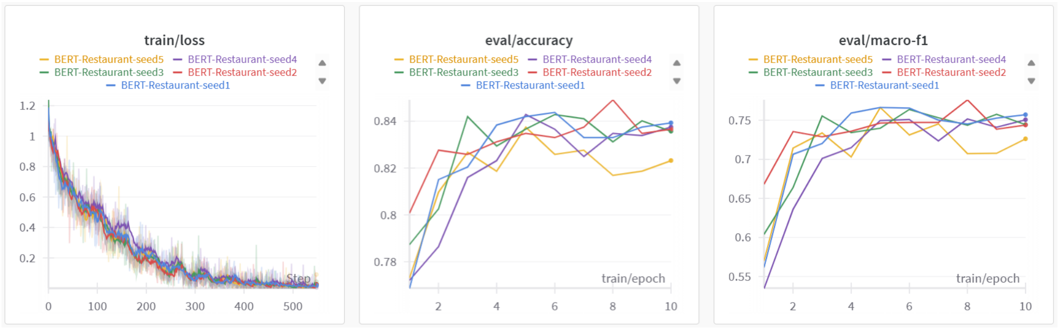

Figure 3: Learing curves of BERT model over Restaurant dataset.

RoBERTa The learning curves over ACL dataset: See Fig. 4

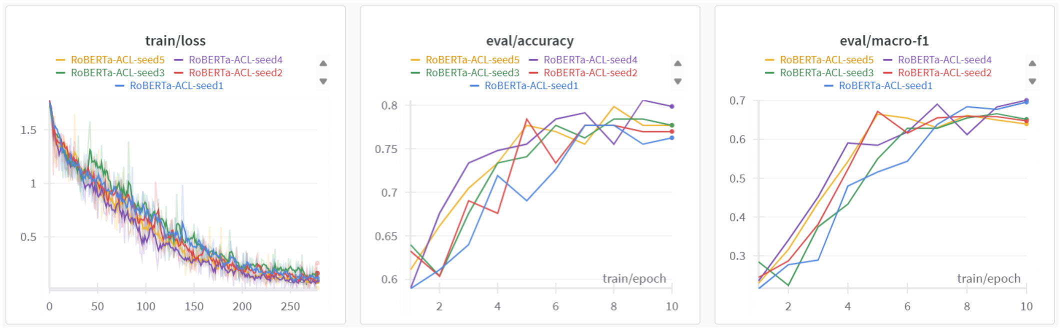

Figure 4: Learing curves of RoBERTa model over ACL dataset.

The learning curves over Agnews dataset: See Fig. 5

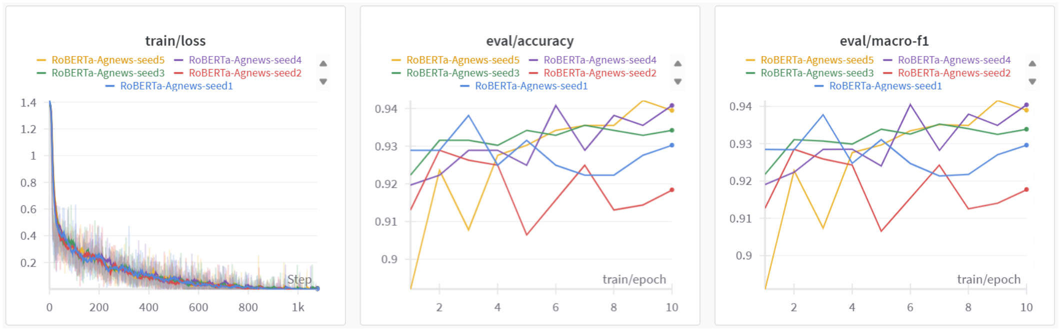

Figure 5: Learing curves of RoBERTa model over Agnews dataset.

The learing curves over Restaurant dataset: See Fig. 6

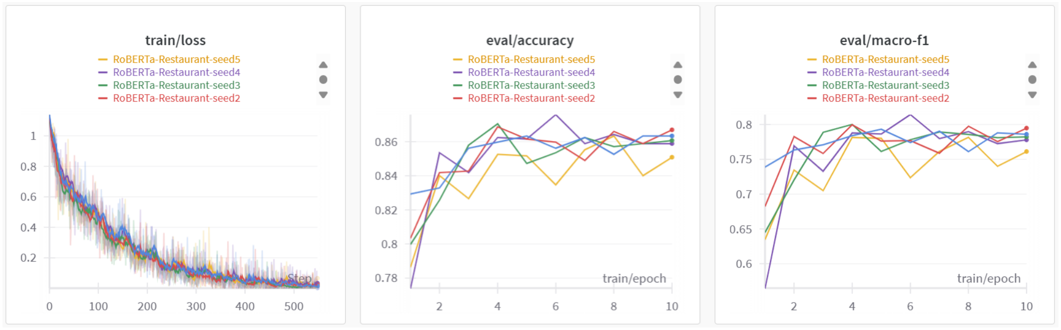

Figure 6: Learing curves of RoBERTa model over Restaurant dataset.

SciBERT The learning curves over ACL dataset: See Fig. 7

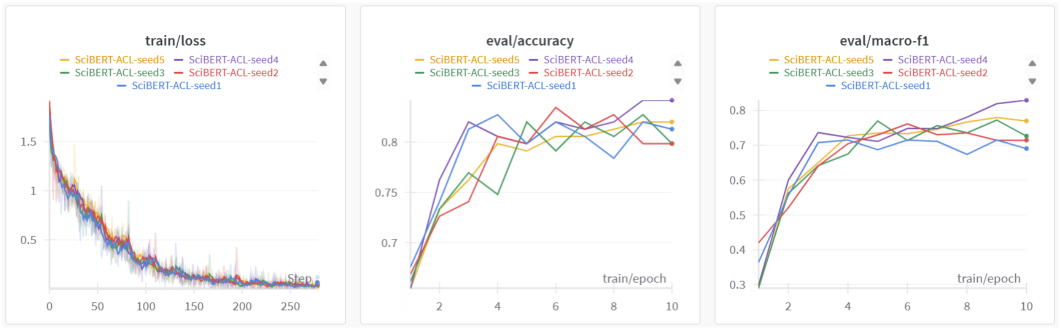

Figure 7: Learing curves of SciBERT model over ACL dataset.

The learning curves over Agnews dataset: See Fig. 8

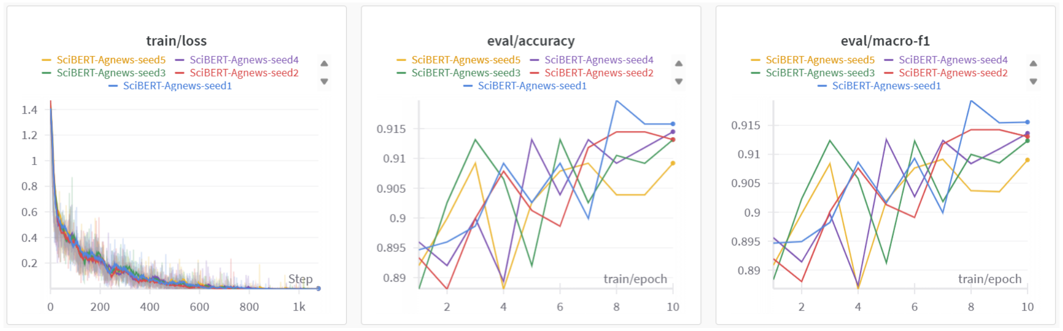

Figure 8: Learing curves of SciBERT model over Agnews dataset.

The learing curves over Restaurant dataset: See Fig. 9

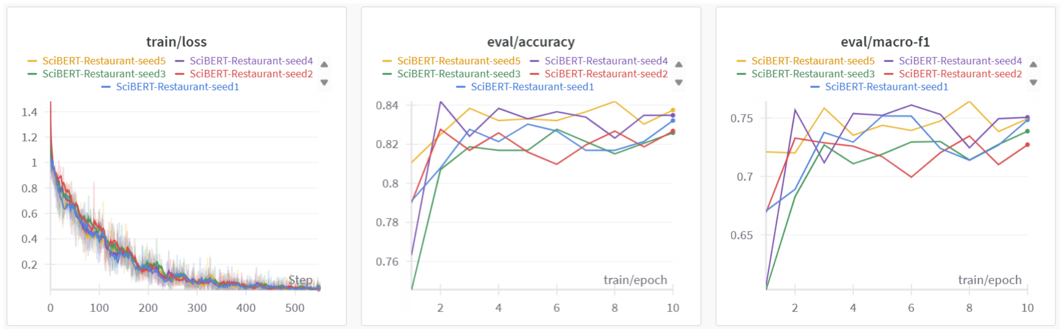

Figure 9: Learing curves of SciBERT model over Restaurant dataset.

1.3 Discussion

We can detect from Tab. 1 and Tab. 2 that the fine-tuned RoBERTa model outperforms the BERT model on all three datasets. The phenomenon presumedly suggests that the RoBERTa model serves as a stronger PLM than the BERT model. It is also obvious that the SciBERT model showcases much better capability on the ACL dataset but performs slightly worse on the other two datasets. I suppose it can be primarily attributed to the discrepancies between the pre-training data of SciBERT and the other two models. It is reported that the SciBERT model concentrates on science literature during the pre-training phase, which corresponds to the theme of ACL dataset to a large extent.

2 Comparisons and analyses of full-parameter fine-tuning and PEFT

In this section, I implement Parameter-Efficient Fine-Tuning (PEFT), more specifically, the Adapter method and the LoRA method (proposed by Houlsby et al. [3] and Hu et al. [4] respectively). The experimental configurations will be presented in Sec. 2.1 and the results will be displayed in Sec. 2.2. I will provide analyses on the GPU memory usage of different fine-tuning methods in Sec. 2.3 and some other discussions in Sec. 2.4.

2.1 Configurations

I leverage the RoBERTa model as the backbone and deploy Adapter and LoRA. For each experiment, I set *learning rate* as 5×10^{-4} in accordance with the small number of trainable parameters in PEFT process. I configure *reduction factor* to 16 for the Adapter method and specify *rank* to 32 for the LoRA method. The unmentioned hyperparameters are kept at their default values. Other configurations remain the same as Sec. 1.1.

2.2 Results

2.2.1 Quantitative results

Here I list the quantitative results of the aforementioned two PEFT methods over the test sets of various datasets.

Accuracy See Tab. 3.

	ACL	Agnews	Restaurant
RoBERTa	$\mathbf{77.70^{\pm 1.35}}$	$\mathbf{93.26^{\pm 0.90}}$	$86.02^{\pm 0.60}$
Adapter	$74.68^{\pm 1.72}$	$92.55^{\pm 0.46}$	$\mathbf{86.35^{\pm 0.14}}$
LoRA	$74.96^{\pm 0.93}$	$92.50^{\pm 0.74}$	$85.62^{\pm 0.30}$

Table 3: The accuracy score (%) of different fine-tuning methods over various datasets.

Macro-F1 See Tab. 4.

	ACL	Agnews	Restaurant
RoBERTa	$\mathbf{66.70^{\pm 2.89}}$	$\mathbf{93.21^{\pm 0.91}}$	$78.05^{\pm 1.24}$
Adapter	$61.63^{\pm 3.21}$	$92.53^{\pm 0.45}$	$\mathbf{78.77^{\pm 0.41}}$
LoRA	$63.21^{\pm 1.03}$	$92.48^{\pm 0.74}$	$77.62^{\pm 0.47}$

Table 4: The macro-f1 score (%) of different fine-tuning methods over various datasets.

2.2.2 Learning curves

I exhibit the learning curves of the aforementioned two PEFT methods over various datasets during training process.

Adapter The learning curves over ACL dataset. See Fig. 10.

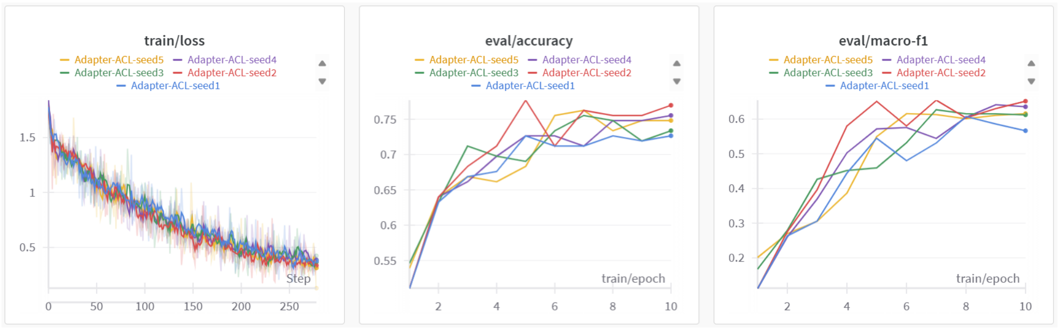

Figure 10: Learing curves of Adapter method over ACL dataset.

The learning curves over Agnews dataset. See Fig. 11.

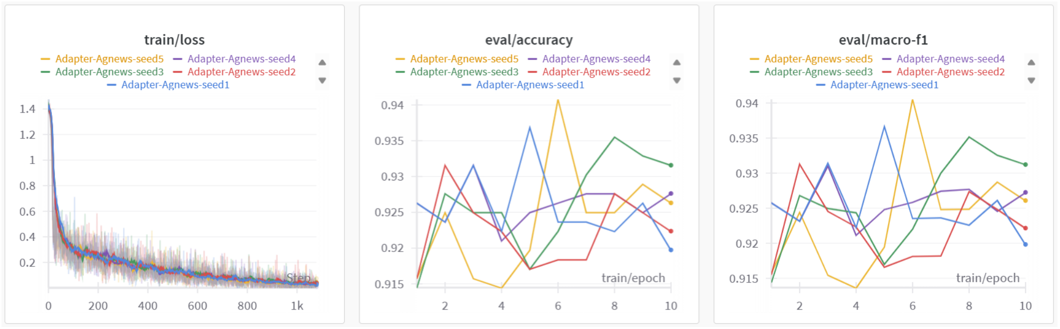

Figure 11: Learing curves of Adapter method over Agnews dataset.

The learning curves over Restaurant dataset. See Fig. 12.

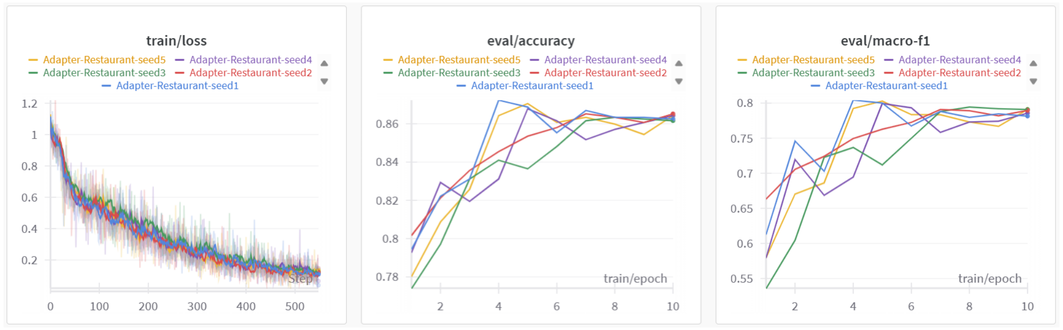

Figure 12: Learing curves of Adapter method over Restaurant dataset.

LoRA The learning curves over ACL dataset. See Fig. 13.

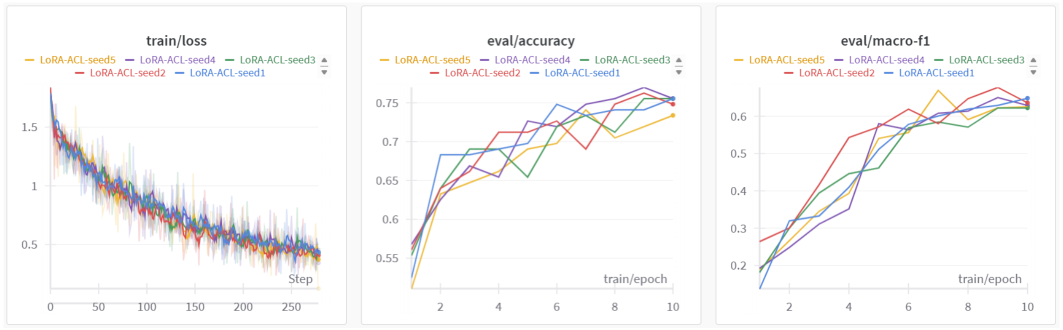

Figure 13: Learing curves of LoRA method over ACL dataset.

The learning curves over Agnews dataset. See Fig. 14.

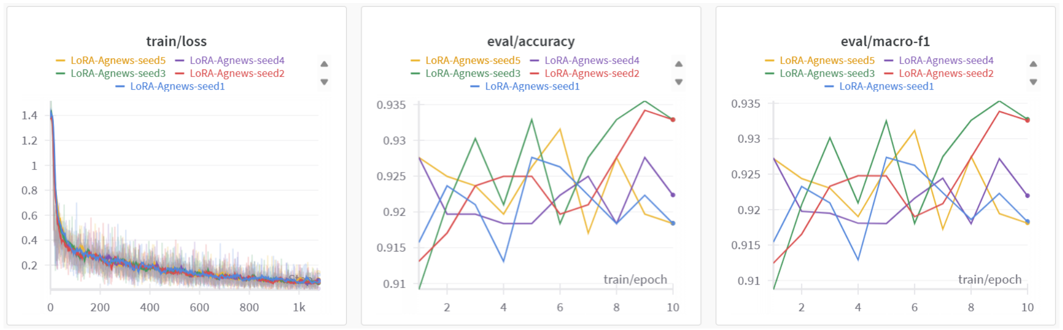

Figure 14: Learing curves of LoRA method over Agnews dataset.

The learning curves over Restaurant dataset. See Fig. 15.

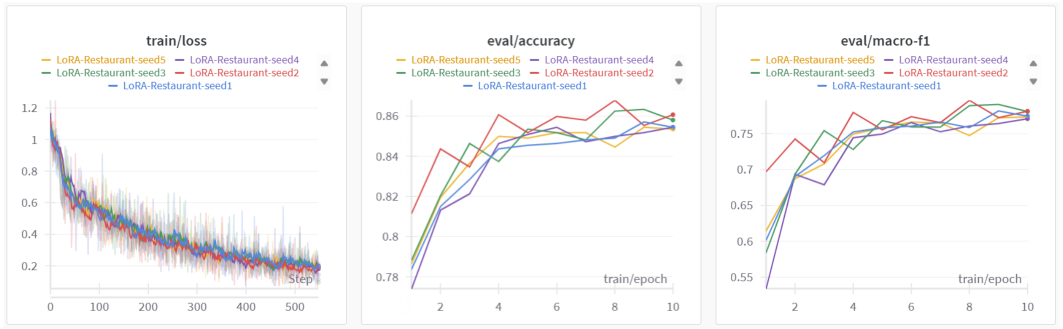

Figure 15: Learing curves of Lora method over Restaurant dataset.

2.3 Analyses of GPU memory usage

In this section, I will conduct fully theoretical analyses on the GPU memory occupation for the full-parameter fine-tuning and the PEFT of the RoBERTa model.

2.3.1 Full-parameter fine-tuning

Initially, I will clarify the notations. Let h denote the hidden size, a denote the number of the attention heads, h' denote the inner hidden size of the feed-forward layers, v denote the vocabulary size, d denote the number of transformer layers, b denote batch size, l denote the length of the input sequence, and N denote the total number of the parameters.
Firstly, consider the memory occupied by the model and the optimizer. Assume the data type we utilize for storing float numbers is FP32, which occupies 4B in memory. For each parameter, there are four float numbers to be saved in the memory: the parameter itself, its gradient, its first-order moment and its second-order moment. The last two items are saved for the sake of the AdamW optimizer. The total memory usage of the model and the optimizer is $16N$B.
Secondly, consider the memory occupied by the intermediate activations stored for gradient computation during the forward propagation phase. The embedding layer does not need intermediate activations, and thus we only need to consider the encoder layers. The detailed computational procedure of one conventional encoder layer is as the following formulas (parameters in bias matrices are ignored):

$$head_i = Dropout\left(Softmax\left(\frac{W_i^Q x(W_i^K x)^T}{\sqrt{d_k}} + mask_{att}\right)\right) W_i^V x. \tag{1}$$

$$x_1 = MultiHead(x) = Concat(head_1, \dots, head_a)W^O \tag{2}$$

$$x_2 = LayerNorm(Dropout(x_1) + x) \tag{3}$$

$$x_3 = W_2 GeLU(W_1 x_2) \tag{4}$$

$$x_4 = LayerNorm(Dropout(x_3) + x_2) \tag{5}$$

As shown in Eq. (1), Eq. (2), Eq. (3), Eq. (4) and Eq. (5), in order to obtain the derivative of W_i^Q, W_i^K, W_i^V, W^O, W_1 and W_2, we primarily need to store the following values:

1. the input $x \in \mathbb{R}^{b\times l\times h}$;
2. the results of $W^K x \in \mathbb{R}^{b\times a\times l\times h/a}$;
3. the results of $W^Q x \in \mathbb{R}^{b\times a\times l\times h/a}$;
4. the results of all $W_i^Q x(W_i^K x)^T \in \mathbb{R}^{b\times a\times l\times l}$
5. the results of $Softmax(\dots) \in \mathbb{R}^{b\times a\times l\times l}$;
6. the first mask matrix derived from the $Dropout$ function in Eq. (1) $mask_1 \in \mathbb{R}^{b\times a\times l\times l}$;
7. the results of $W^V x \in \mathbb{R}^{b\times a\times l\times h/a}$;
8. the results of $Concat(\dots) \in \mathbb{R}^{b\times l\times h}$;
9. the second mask matrix derived from the $Dropout$ function in Eq. (3) $mask_2 \in \mathbb{R}^{b\times l\times h}$;
10. the input of the $LayerNorm$ function in Eq. (3) $Dropout(x_1) + x \in \mathbb{R}^{b\times l\times h}$
11. the results of $x_2 \in \mathbb{R}^{b\times l\times h}$
12. the results of $W_1 x_2 \in \mathbb{R}^{b\times l\times h'}$
13. the results of $GeLU(W_1 x_2) \in \mathbb{R}^{b\times l\times h'}$
14. the third mask matrix derived from the $Dropout$ function in Eq. (5) $mask_3 \in \mathbb{R}^{b\times l\times h}$;
15. the input of the $LayerNorm$ function in Eq. (5) $Dropout(x_3) + x_2 \in \mathbb{R}^{b\times l\times h}$

Summing up all the terms together and multiplying the number of encoder layers d, we can figure out the the (estimated) total memory of the intermediate activations is:

$$d \times (34blh + 9bal^2 + 8blh') \text{ B}$$

The ultimate expression for the full-parameter fine-tuning memory is:

$$16N + d \times (34blh + 9bal^2 + 8blh') \text{ B}$$

In the meanwhile, we have an estimated equation between N and other hyperparameters:

$$N = vh + d \times (4h^2 + 2hh' + 4h)$$

To estimate the memory, we can assign a set of appropriate values to h, h', v, d, and a under the constraint $N \approx 3 \times 10^9$. Hence, I choose $v = 5 \times 10^4$, $h = 1920$, $h' = 7680$, $d = 64$, $a = 32$. Substitute the numbers into the expression, we can obtain the estimated GPU memory usage:

$$4.8 \times 10^4 + 1.84 \times 10^{-2} bl^2 + 8.11bl \text{ MiB}$$

where b and l depend on specific experiments and can vary in broad ranges.

2.3.2 PEFT

Here I will merely analyze the GPU memory of the Adapter method. Let $h_{bottleneck}$ denote the bottleneck size of the adapters and $N_{adapter}$ denote the number of parameters in adapters. Akin to Sec. 2.3.1, we can calculate the memory of the model and the optimizer first. The overall occupation is:

$$4N + 16N_{adapter} \text{ B}$$

Then we need to consider the intermediate activations stored for adapter layers. The formula of the adapter layer is as following:

$$x_{out} = W_{up} Swish(W_{down} x_{in}) + x_{in} \tag{6}$$

In order to calculate the derivative of W_{down} and W_{up}. We need to store the following values:

1. the values of $X_{in} \in \mathbb{R}^{b \times l \times h}$
2. the results of $W_{down} x \in \mathbb{R}^{b \times l \times h_{adapter}}$
3. the results of $Swish(W_{down} x) \in \mathbb{R}^{b \times l \times h_{adapter}}$

Summing up all the terms above, multiplying the number of encoder layers as well as adding the model memory and the optimizer memory, we obtain:

$$4N + 16N_{adapter} + 2d \times (4blh + 8blh_{adapter}) \text{ B}$$

Moreover, there is an equation between $N_a dapter$ and other hyperparameters:

$$N_{adapter} = 4dhh_{adapter} + 4dh$$

If we inherit the hyperparameters we prepend in Sec. 2.3.1 and further assign 120 to $h_{adapter}$, in other words, the reduction factor is 16. We can get a new expression concerning b and l:

$$1.3 \times 10^4 + 1.11bl \text{ MiB}$$

Even though we cannot determine the specific b and l, we can tell that the Adapter method can save at least $3/4$ of the memory.

2.4 Discussion

Firstly, from Tab. 3 and Tab. 4, we can evidently observe that both PEFT methods performs (relatively) well on all three datasets. On the Restaurant dataset, the Adapter method even outperforms full-parameter fine-tuning. In the meanwhile, comparing Fig. 12 and Fig. 6 (or any other similar pairs), it is obvious that PEFT methods usually converge worse,i.e. reach a higher loss plateau, on the training set. The phenomena reveal the superior generalization capability of PEFT methods. My hasty hypothesis is that due to the exceeding flexibility of the model, full-parameter fine-tuning tends to overfit the model to the training data and to some extent undermines the inherent language capability of the PLMs. Secondly, it is noteworthy that in most of the cases, the evaluated metrics of PEFT methods on the test set display smaller standard deviations than full-parameter fine-tuning. It suggests that PEFT methods are more steady in practice. Thirdly, in addition to Adapter method required by the assignment, I implement LoRA method for comparison. From the results in Tab. 3 and Tab. 4, we can observe that LoRA method outperforms Adapter method on ACL dataset but lags behind on the other two datasets. This gives us the insights that there are intrinsic discrepancies among different PEFT methods, leading to diverse choices of PEFT methods in the context of different tasks.

3 Conclusion

In this concise exploration, we delve into the methodologies of full-parameter fine-tuning and PEFT. These approaches exemplify the significant power of the pre-training and fine-tuning paradigm, highlighting its capability to adapt large-scale pre-trained models to specific tasks with relatively low computational costs. In comparison to full-parameter fine-tuning, PEFT demonstrates several notable advantages, particularly in scenarios where computational resources are limited. By adjusting or adding only a small number of parameters, PEFT effectively mitigates the risk of overfitting while significantly reducing training time and computational requirements. As model sizes continue to grow and computational resources remain a constraint in this era, the incredible significance of PEFT will undoubtedly become even more pronounced.

References

[1] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text, 2019. URL `https://arxiv.org/abs/1903.10676`. 1

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL `https://arxiv.org/abs/1810.04805`. 1

[3] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019. URL `https://arxiv.org/abs/1902.00751`. 4

[4] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL `https://arxiv.org/abs/2106.09685`. 4

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. URL `https://arxiv.org/abs/1907.11692`. 1