Rasoul Zahedifar

Student Number: 401210215

Email: rasoul.zahedifar@gmail.com

Instructors Team:

Professor M. Soleymani Baghshah soleymani@sharif.edu

Professor M. H. Rohban rohban@sharif.edu

Professor E. Asgari asgari@berkeley.edu

Head Teaching Assistant:

M. R. Fereydooni mrezafereydooni@gmail.com

Assignment-in-charge Teaching Assistant:

M. A. Sadrayi Javaheri mohammadalisadraei@gmail.com

Large Language Models Course, Assignment 1: Parameter-Efficient Fine-Tuning

Fall 2023

In this report, we evaluate various Parameter-Efficient Fine-Tuning (PEFT) methods for fine-tunning on a large language model. We specifically investigate full fine-tuning, soft prompting, adapters, and Lora, with T5-small serving as our baseline model. Our analysis focuses on crucial aspects such as the resulting model sizes, their performance during test time and effectiveness in mitigating overfitting risks. Our data is IMDB dataset and the task is sentiment analysis. In the following, we show the performance of the model on each method in Table 1.

Table 1. The Performance of Model

Index	**Notebook Num**	**Fine-tunning Method**	**Description of Variable**	**Value of Variable**
1	1	Full fine-tunning	Best Model Performance	90.1 %
2	2	Soft-prompt tunning	Best Model Performance $@num\_soft\_token = 10$	87.1 %
3	2	OpenDelta	Best Model Performance $@num\_soft\_token = 1$	85.0 %
4	2	OpenDelta	Best Model Performance $@num\_soft\_token = 10$	86.5 %
5	3	Adapter	Best Model Performance $@bottleneck\_size = 8$	90.5 %
6	4	Adapter Hub	Best Model Performance $@bottleneck\_size = 8$	90.1 %
7	4	Adapter Hub	Best Model Performance $@bottleneck\_size = 1$	89.1 %
8	5	Lora	Best Model Performance $@rank = 8$	89.1 %
9	5	PEFT	Best Model Performance $@rank = 1$	88.5 %
10	5	PEFT	Best Model Performance $@rank = 8$	88.7 %

As we can see in Table 1, the Adapter model exhibits the highest performance, achieving an accuracy of 90.5%. Following closely are the full fine-tuning and Adapter Hub, characterized by a Bottleneck size of 8, which attains the second-best performance at 90.1%. The Lora model, in conjunction with Adapter Hub (Bottleneck size=1), follows with a performance level of 89.1%.

Further examination reveals that the Lora model, implemented using the PEFT library with ranks 1 and 8, records accuracies of 88.5% and 88.7%, respectively. In contrast, the soft-prompt tuning method demonstrates comparatively lower performance, registering an accuracy of 87.1%. Specifically, when utilizing the OpenDelta Library with soft token counts of 1 and 10, the corresponding accuracies are 85.0% and 86.5%. These findings provide a detailed comparative analysis of the performance of the evaluated fine-tunning methods under different configurations.

As for why the full fine-tunning did not record the best performance, it is due to the catastrophic forgetting phenomenon which may have led to the overfitting as a common circumstance in full fine-tunning methods. The larger the dataset, the more we can avoid this situation. Another thing to note is the performance of soft prompt fine-tunning. Despite the fact that it is recorded among the weaker methods, considering the small number of its trainable parameters, it can even be construed as a competitive method. It only needs to train 512 times the number of soft tokens, take the least amount for the number of soft tokens, namely 1, the performance of the model is only around 5% behind the full fin-tunning as the baseline. This trend can be leveraged to even less than 3% behind the baseline, only using number of soft tokens equals to 10.

Interestingly, Adapter come out as the best method. We assume this is due to the characteristics of the adapters that avoid overfitting, where bottleneck adapters can act as a form of regularization by limiting the number of parameters that are updated during training. On the other hand, bottleneck adapters typically focus on modifying or fine-tuning a small subset of parameters in a pre-trained model. This allows the model to retain most of the knowledge from the pre-training phase, which can be highly beneficial if the pre-trained features are transferable to the new task.

Moreover, the improvement switching from rank 1 to 8 shows that we can have better performance with more trainable parameters, but the considerable fact is that even with the lowest amount of rank, namely 1, the results are so competitive with lower amounts of trainable parameters compared to the baseline with full fine-tunning.

In the next section, we are going to analyze these methods on the number of their parameters. First, we present Table. 2, where the number of parameters of the non-library methods are given.

Table 2. The Number of Parameters

Index	**Notebook Num**	**Fine-tunning Method**	**Description of Variable**	**Value of Variable**
1	1	Full fine-tunning	Number of parameters	60,506,624
2	2	Soft-prompt tunning	Number of parameters	60,511,744
3	3	Adapter	Number of parameters	60,611,168
4	5	Lora	Number of parameters	60,801,536

Considering Table 2 which provides valuable insights into their respective architectural complexities, the Lora model, occupying the first position in the comparison, is characterized by a parameter count of 60,801,536. This is followed by the Adapter model, positioned second, with the number of parameters equal to 60,611,168. In the third position, the Soft-prompt tunning model distinguishes itself with a parameter count of 60,511,744. Lastly, the Full fine-tunning model, placed fifth in the ranking, exhibits the number of parameters as 60,506,624.

In the last part, we analyze the number of parameters introduced with Library methods and investigate whether, considering the hyperparameters we added to them, their number of parameters are in align with their non-library counterparts.

First, we start with Soft-prompt tunning using OpenDelta. It can be argued that the number of added parameters due to the introducing number of soft token yields as:

$$Num\ of\ added\ parameters = E \times S$$

where, E is the embedding size which is 512 for T5 model and S is the number of soft tokens. That is due to the fact we will have embedding size vector for each soft token, as a result, embedding size times the number of soft tokens yields to the number of added parameters. Therefore:

$$Num\ of\ added\ parameters = E \times S = 512 \times 1 = 512$$

This results in the following equation where the number of parameters is calculated as:

$$\begin{aligned} &Num\ of\ parameters \\ &\qquad = Num\ of\ added\ parameters + Num\ of\ original\ model\ parameters \\ &\qquad = 512 + 60{,}506{,}624 = 60{,}507{,}136 \end{aligned}$$

which is in accordance with the result of the notebook. Similarly, we can have the same trend for the number of soft token equals to 10 as:

$$Num\ of\ added\ \ parameters = E \times S = 512 \times 10 = 5{,}120$$

and we continue as:

$$\begin{aligned} &Num\ of\ parameters \\ &\qquad = Num\ of\ added\ parameters + Num\ of\ original\ model\ parameters \\ &\qquad = 5{,}120 + 60{,}506{,}624 = 60{,}511{,}744 \end{aligned}$$

which is, too, comparable with the result of the notebook.

In the next step, we investigate the same analysis for the adapter section where the number of added parameters can be calculated as:

$$Num\ of\ added\ parameters = (N_e + N_d) \times \big(E \times (B + 1) + B \times (E + 1)\big)$$

where, N_e and N_d are the number of encoder and decoder blocks respectively, E is the embedding size which is 512 for T5 model, and B is the bottleneck size. Also, note that the 1 added to the dimension is represented as the bias vector. About how to obtain the formula, we know that there is an up projection and down projection between embedding size and bottleneck size along with their correspondence biases, which should be multiplied to the number of times we have this trend, namely, the sum of encoder and decoder numbers. Therefore, with bottleneck size equals to 8, we have:

$$\begin{aligned} Num\ of\ added\ parameters &= (N_e + N_d) \times \big(E \times (B+1) + B \times (E+1)\big) \\ &= (6+6) \times \big(512 \times (8+1) + 8 \times (512+1)\big) = 104{,}544 \end{aligned}$$

and we can get the number of parameters as follows:

$$\begin{aligned} &Num\ of\ parameters \\ &\qquad = Num\ of\ added\ parameters + Num\ of\ original\ model\ parameters \\ &\qquad = 104{,}544 + 60{,}506{,}624 = 60{,}611{,}168 \end{aligned}$$

which is compatible with the result of the notebook. Also, the same argument is true for the bottleneck size of 1 as:

$$\begin{aligned} Num\ of\ added\ parameters &= (N_e + N_d) \times \big(E \times (B+1) + B \times (E+1)\big) \\ &= (6+6) \times \big(512 \times (1+1) + 1 \times (512+1)\big) = 18{,}444 \end{aligned}$$

and it yields to:

$$\begin{aligned} &Num\ of\ parameters \\ &\qquad = Num\ of\ added\ parameters + Num\ of\ original\ model\ parameters \\ &\qquad = 18{,}444 + 60{,}506{,}624 = 60{,}525{,}068 \end{aligned}$$

which is the same as the result of the notebook.

In the last part, we analyze the Lora method, where the number of parameters is achieved as:

$$Num\ of\ added\ params = (N_s + N_c) \times n \times (E \times r + r \times E)$$

where, N_s is the number of self-attention blocks (equals to $N_e + N_d$) and N_c is the number of cross attention blocks (equals to N_d), n is the number of trainable blocks, E is the embedding size which is 512 for T5 model and r is the rank. As for how to get the above formula, we should consider that there is no bias vector based on Lora paper. So, similar to the previous reasoning in adapter, the number of up projection and down projection parameters is given by the multiplying the embedding size to rank. And we have this trend for the number of trainable blocks (which is 2 because of query and value) times number of sum of self-attention and cross-attention blocks. As a result, with rank equals to 8, we get:

$$\begin{aligned} Num\ of\ added\ params &= (N_s + N_c) \times n \times (E \times r + r \times E) \\ &= (12+6) \times 2 \times (512 \times 8 + 8 \times 512) = 294{,}912 \end{aligned}$$

and the total number of parameters can be obtained as:

$$\begin{aligned} &Num\ of\ parameters \\ &\quad = Num\ of\ added\ parameters + Num\ of\ original\ model\ parameters \\ &\quad = 294{,}912 + 60{,}506{,}624 = 60{,}801{,}536 \end{aligned}$$

which is accordant with the result of the given notebook. Moreover, considering rank equals to 1, it yields:

$$\begin{aligned} Num\ of\ added\ parameters &= (N_s + N_c) \times n \times (E \times r + r \times E) \\ &= (12 + 6) \times 2 \times (512 \times 1 + 1 \times 512) = 36{,}864 \end{aligned}$$

and we obtain the total number of parameters as:

$$\begin{aligned} &Num\ of\ parameters \\ &\quad = Num\ of\ added\ parameters + Num\ of\ original\ model\ parameters \\ &\quad = 36{,}864 + 60{,}506{,}624 = 60{,}543{,}488 \end{aligned}$$

which is the same as what we achieved in the notebook.

Should you have any concern or question related to this assignment, please feel free to reach out to the writer at rasoul.zahedifar75@gmail.com at your convenience.