Implementation and Experimental Analysis of LoRA: Low-Rank Adaptation of Large Language Models Report

LIU ZhiXian, TIAN XiaoWen, ZHANG WeiXuan, ZHANG Zhe

I. Abstract

This paper presents our implementation and analysis of the Low-Rank Adaptation (LoRA) method for fine-tuning large language models. We compare LoRA with traditional fine-tuning and adapter methods on the RoBERTa-Large model using the Semantic Textual Similarity Benchmark (STSB). Our experiments demonstrate that LoRA achieves 98.1% of full fine-tuning performance while using only 0.08% of the trainable parameters and reducing training time by 48%. We also analyze the impact of applying LoRA to different attention matrices and identify optimal configurations. Our findings confirm that LoRA offers an efficient and effective approach for adapting large pre-trained models to downstream tasks with minimal computational resources.

II. Introduction

Our team reproduced the LoRA (Low-Rank Adaptation) method proposed in the paper "LoRA: Low-Rank Adaptation of Large Language Models" [1], which introduces an efficient and parameter-saving approach for fine-tuning large-scale pre-trained models. LoRA freezes the original weights of the model and injects trainable low-rank decomposition matrices into each layer, thereby significantly reducing the number of trainable parameters and GPU memory usage during adaptation.

For our experiments, we selected RoBERTa-Large (335M parameters) as the base pre-trained language model. This model was adapted to a range of downstream tasks including sequence classification, label classification, and question answering, using the LoRA framework [2]. We fine-tuned the model on the STSB (Semantic Textual Similarity Benchmark) dataset, which requires predicting the degree of semantic similarity between two sentences, with target scores ranging from 0 to 5 [3].

To evaluate the effectiveness of LoRA-based fine-tuning, we adopted the Pearson correlation coefficient as the primary metric, quantifying the linear correlation between model predictions and the truth similarity scores.

III. Experiment

This experiment aims to evaluate the efficiency, performance, and stability of LoRA and other fine-tuning methods.

A. *Experiment Configuration*

The experiments were conducted on NVIDIA RTX 3070Ti(8GB), using Pytorch 2.0.0 with CUDA 11.7. Full-tuning modules were implemented via HuggingFace PEFT v0.15.0 and Adapters v1.1.0.

The experiment adopts a standardized evaluation framework, all methods shared identical optimization settings:

- BaseModel: Roberta-large
- Optimizer: AdamW
- Learning rate: 1e-4
- Batch size: 16
- Training epochs: 5
- LoRA: Rank $r = 3$, scaling factor $\alpha = 16$, injected into all Q/K/V/O layers
- AdapterH: Bottleneck dimension $d = 3$, GELU activation, Modified FFN output projection

This Configuration resulted in LoRA 0.08% (0.29M) vs Adapter 0.06% (0.20M) of full fine-tuning parameters.

B. *Training Efficiency Analysis*

LoRA and Adapter achieved 53.1% (172 ± 3.2s/epoch) and 54.5% (148 ± 2.8s/epoch) time reduction compared to full fine-tuning (325 ± 5.1s/epoch). Backpropagation time exhibited linear correlation with parameter counts, confirming the theoretical expectation of reducing the complexity of computation by LoRA.

C. *Model Stability and Performance Evaluation*

The STS-B dataset was selected as the evaluation benchmark for performance and stability analysis. Pearson correlation coefficient and variation coefficient were used to quantify the performance and stability of fine-tuning methods. The experiments were independently run 5 times with random seeds.

Focusing on the last epoch, LoRA achieved Pearson=0.91 ± 0.008 and Adapter achieved Pearson=0.82 ± 0.014, which accounted for 98.1% and 88.9% of full fine tuning respectively. The 9.2% performance gap between LoRA and Adapter indicates LoRA's superior capacity.

During the first epoch, Adapter displayed high instability (CV=50.01%) compared to LoRA's performance (CV=6.24%). By the final epoch, LoRA maintained CV=0.51% while Adapter remained CV=2.22%.

D. *LoRA on different attention matrices*

We have tested LoRA implementation on Q, V, K and O four matrices and their combinations using ranks 1, 2, 4, 8, and 32. The result showed that under STS-B dataset and roberta-large model, targeting specific matrices significantly impacts performance. When using individual matrices, V and O achieved the highest Pearson correlation of 0.912 ± 0.001, while K and Q performed poorly (Pearson 0.86). And as deduced, the combination of V and O get the best prediction result around 0.917.

E. *Summary*

LoRA achieved 98% of baseline performance while reducing training time by 48%, maintaining stable performance. Therefore LoRA is an optimal choice for resource-efficient model adaptation, particularly in production environments where reliability and predictability are essential. Also different attention matrices have great influence on the LoRA fine tuning method, in this task we suggest using O and V metrics.

IV. Challenges

A. *Loss Function Selection*

In our project, we define specific loss functions according to the goal of each dataset.

In the case of classification datasets "snli" and "ag_news", we utilize Cross Entropy Loss, which can directly optimize the probability distribution of targets by penalizing predictions of incorrect classes.

When it comes to the dataset "stsb", which predicts the similarity score in a continuous range from 0 to 5, we select MSE Loss to minimize the deviation between ground truth and prediction.

B. *Data Pre-processing*

We encountered several challenges when loading our datasets.

- Missing data:

In the "snli" [4] dataset, there are around 700 samples with missing labels, which are assigned with value -1. In the pre-processing stage, we filter out these data to ensure the validity of the dataset:

```
dataset["train"] = dataset["train"].filter(
lambda x: x["label"] in [0, 1, 2])
dataset["validation"] = dataset["validation"].
filter(lambda x: x["label"] in [0, 1, 2])
```

- Tokenization:

Take the "stsb" dataset as an example, during initial training experiments, our model failed to optimize parameters, the Pearson correlation remained around 0.07 after 10 epochs. After inspecting the code, we realized that in the pre-processing step, we removed all the columns in the training set, leading to the loss of supervision signals.

```
tokenized_datasets = dataset.map(
    preprocess_function,
    batched=True,
    remove_columns=dataset["train"].
        column_name
)
```

Subsequently, we attempted to manually restore the label column, but the Pearson correlation remained unchanged since batch processing caused index mismatches.

```
tokenized_datasets = tokenized_datasets.map(
    lambda examples, indices: {"labels":
    dataset["train"][indices]["label"]},
    with_indices=True,
    batched=True,
)
```

Finally, we restrict the removed columns to textual ones while preserving the label column, and this correlation ensures a proper backpropagation process, the model successfully converges to 0.88 after 5 epochs, greatly improved compared to the initial value 0.07.

```
tokenized_datasets = dataset.map(
    preprocess_function,
    batched=True,
    remove_columns=["sentence1", "sentence2"])
```

This case highlights that it is necessary to explicitly retain supervision signals during the pre-processing stage.

C. *Parameter freezing problem*

In the experiments on applying LoRA in different attention matrices, we have to test LoRA implementation on the query, key, value, and output matrices. As imaginary, the parameter should grow proportionally with the number of matrices we decided to implement, however, the number of parameters of each experiment is around 600000 and changes little between the different settings. After checking everything related to the parameters config, We discovered that the issue arose because a classification output layer was automatically appended to the end of the model to generate the correct number of labels, and by default, the parameters in this layer were not frozen. Once we freeze the parameters, the parameters grow correctly.

D. *Full parameter finetuning*

At the beginning we set the learning rate of full parameter fine tuning to be 2*e-4, and find after one epoch, we encountered a problematic training loss scenario. Specifically, the loss did not decrease as expected; instead, it remained stagnant. Meanwhile, the Pearson correlation score fluctuated significantly around zero. After conducting research online, we discovered that others had faced similar issues and we found the correct answer in one comment, the learning rate 2*e-4 is excessively high for the model to train effectively when all parameters are unfrozen. Finally after setting the training learning rate into 1*e-5, the training process then functioned as intended. We guess the problem is due to the pre-trained parameters changing significantly in large training steps, and thus undermining the prediction ability.

E. *Adapter Parameter*

To ensure the fairness of the experiment, we need to make the parameter count of LoRA and Adapter the same. Initially, we set both LoRA and Adapter with parameter rank=3, resulting in vastly different parameter counts: 0.3M for LoRA and 3M for Adapter.

Upon investigation, we found that the parameter count of Adapter is determined by the reduction factor, which indicates the degree of adapter dimensionality reduction, where $r = d/\text{rank}$. Therefore:

$$\text{Adapter Parameters} = 2 \times d \times \frac{r}{d} + \frac{r}{d} + d = 2dr + d + \frac{r}{d} \tag{1}$$

$$\text{LoRA Parameters} = 2 \times d \times r \tag{2}$$

If we want to set the same parameter count between LoRA and Adapter, we should set:

$$\text{Adapter Parameters} \approx \text{LoRA Parameters} \tag{3}$$

$$\Rightarrow \frac{r_{\text{Adapter}}}{2d^2} \approx 2dr_{\text{LoRA}} \tag{4}$$

$$r_{\text{Adapter}} = \frac{d}{r_{\text{LoRA}}} \tag{5}$$

For RoBERTa-large with hidden dimension $d = 1024$ and LoRA rank $r_{\text{LoRA}} = 3$, we should set the reduction factor $r_{\text{Adapter}} = 1024/3 \approx 341$, instead of 3.

V. Conclusion

Compared to full-parameter fine-tuning, LoRA achieves similar performance while significantly reducing the number of parameters. The training results are also highly robust. For 6k training data in STSB, a small rank value in LoRA is usually enough to produce good results. However, unlike this paper's practice of optimizing the query layer and the key layer, our experiments show that applying LoRA to the output layer and the value layer works better. Overall, LoRA proved to be an excellent method of fine tuning.

References

[1] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," arXiv preprint arXiv:2106.09685, 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[2] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," arXiv preprint arXiv:1907.11692, 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[3] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "SemEval-2017 Task 1: Semantic Textual Similarity – Multilingual and Cross-lingual Focused Evaluation," in Proc. 11th Int. Workshop on Semantic Evaluation (SemEval-2017), Vancouver, Canada, Aug. 2017, pp. 1–14. [Online]. Available: https://aclanthology.org/S17-2001/

[4] Bowman, S. R., Angeli, G., Potts, C., & Manning, C. D. (2015). A large annotated corpus for learning natural language inference. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), 632–642. [Online]. Available: https://aclanthology.org/D15-1075