Reimplementation of LoRA: Low-Rank Adaptation of Large Language Models

Alicia Chu, Rayhan Khanna, Emma Li, Sanjana Nandi, Tejal Nair

https://github.com/tn292/cs5782_final_project

1. Introduction

As models have become larger in recent years, fine-tuning becomes infeasible and expensive. *LoRA: Low-Rank Adaptation of Large Language Models*, a framework proposed by Edward Hu et al., proposes to freeze the pre-trained model weights and inject trainable low-rank matrices into key layers of the Transformer architecture, which reduces the number of trainable parameters by 10,000x and the GPU memory requirement by 3x compared to full fine-tuning.

Key contributions of the paper include:

- Demonstrating that low-rank adapters achieve comparable or better performance than full fine-tuning on a suite of NLP tasks (e.g., GLUE benchmarks).
- Showing that small ranks (even $r = 1, 2$) suffice for high accuracy, and that adapting multiple attention matrices improves results.

2. Chosen Result

We aimed to reproduce the downstream task performance of LoRA using RoBERTa-base on the GLUE benchmark specifically on these tasks: SST-2, QQP, and MRPC (shown in Table 2 in the paper). This is significant because the results show that LoRA achieves comparable or better performance than full-fine tuning while training substantially less parameters.

Additionally, we replicated Table 6, which evaluates LoRA's effectiveness on WikiSQL and MultiNLI using GPT-3. These results highlight two key insights: (1) Even small ranks (e.g., $r = 1$ or 2) achieve performance comparable to high ranks ($r = 64$), and (2) Adapting more attention weights (e.g., W_q, W_k, W_v, W_o) generally leads to better performance, confirming the efficiency and flexibility of low-rank adaptation. Although in our implementation we used RoBERTa-base instead of GPT-3.

3. Methodology

To reproduce the results from the LoRA paper, we reimplemented their training pipeline using the Hugging Face Transformers library. Our goal was to match the reported performances in Tables 2 and 6 of the paper. We focused our implementation on adapting the RoBERTa-base model, using LoRA modules injected into the attention projection layers of the transformer.

3.1. Model Architecture

We used the RoBERTa-base architecture as a backbone in line with the original LoRA paper. The classification heads remained unchanged from their Hugging Face implementations. The primary architectural modification was the injection of LoRA modules into selected linear layers within the self-attention mechanism. This was achieved by replacing specific weight matrices (query, value, and optionally key and MLP) with LoRA-adapted modules. These modules consist of a trainable low-rank decomposition $W + \Delta W$ where $\Delta W = BA$ with $A \in \mathbb{R}^{r \times d}, B \in \mathbb{R}^{d \times r}$ and $r << d$. We implemented this logic in transformers.py in the function patch_model_with_lora, which recursively locates and replaces the target layers with custom LoRALinear modules. Each LoRA layer includes optional dropout and scales the residual via the hyperparameter α as described in the original paper.

3.2. Dataset and Tokenization

We trained and evaluated our models on multiple tasks from the GLUE benchmark using the datasets library. Tokenization was handled by Hugging Face's RobertaTokenizer, with inputs truncated or padded to a maximum sequence length of 128 tokens. This fixed sequence length was chosen to align with the experimental setup used in the LoRA paper for a fair comparison.

3.3. Optimization and Training Setup

We followed the original paper's design for our training runs. All models were trained using Hugging Face's Trainer class with the Adam optimizer with betas (0.9, 0.99), epsilon 1e-8, and a linear learning rate decay schedule with a warm-up ratio of 6%—this was done through our custom_optimizer function in train_lora.py.

As outlined in the paper's Appendix A, a fixed batch size was used for each task we trained on. However, due to computational constraints, we limited the number of training epochs across all tasks to 3. While this deviates from the longer training schedules used in the original paper, it still allows us to capture relative performance and trends.

3.4. Logging and Evaluation

Per-epoch training losses, validation losses, and evaluation metrics were recorded and saved in CSV files within task-specific output directories. Final performance metrics, including training accuracy, precision, recall, and F1-score, were printed at the end of each training session. All experiments were seeded for reproducibility. We additionally automated evaluation over multiple hyperparameter settings using a sweep script (sweep.py) that repeats experiments across five random seeds, and the best epoch for each run was selected based on validation performance. Final results were aggregated by reporting the median accuracy and standard deviation across these five runs, consistent with the evaluation strategy for Table 2 in the original paper.

4. Results

4.1. Reproducing Paper's Table 2 Results

A comparison of our implementation to the true LoRA implementation is provided in sections 6.1 and 6.2 as Table 1 and Fig. 1, as well as the specification of hyperparameters used in Table 2.

While our results are slightly lower than those reported in the original paper, the discrepancies can be attributed to several key differences in training configuration. Most significantly, we limited all tasks to only 3 training epochs due to computational constraints. In contrast, the original paper trained SST-2 for 60 epochs, QQP for 25, and MRPC for 30 epochs. This reduction in training time likely impacted the convergence and final performance of our models.

Additionally, we uniformly set the maximum input sequence length to 128 tokens for all tasks, whereas the original implementation used up to 512 tokens. This limitation stemmed from a challenge with memory constraints on our machines, especially when running multiple experiments or using larger batch sizes. Reducing the sequence length helped alleviate these issues, but may have affected performance on tasks like QQP and MRPC, where relevant contextual information may span longer input sequences.

Despite these modifications, our results demonstrate that LoRA can achieve competitive performance with limited training and shorter input lengths. This validates that LoRA is a highly parameter-efficient fine-tuning method capable of strong generalization even under constrained resources.

4.2. Reproducing Paper's Table 6 Results

While the paper evaluated on GPT-3, we conducted our experiments using RoBERTa-base due to resource constraints. As a result, our absolute accuracies are lower but the trends in performance across ranks and weight types closely follow the original findings.

Our experiments replicated the paper's setup by sweeping over ranks $[1, 2, 4, 8, 64]$ and injecting LoRA into different combinations of attention projection matrices. Due to time constraints, we ran each setting across only one random seed, instead of averaging over 5 random seeds, selected the best-performing epoch by validation accuracy, and reported the accuracy.

Despite using a smaller model and fewer epochs, our results, in section 6.1 labeled as Table 3, confirm the paper's key insight: applying LoRA to multiple attention weights improves performance, and increasing the rank, up to a point, generally leads to better accuracy.

4.3. Analysis

Our results reaffirm the paper: LoRA enables efficient fine-tuning of large models with a significant reduction in trainable parameters while maintaining competitive task performance. Despite limiting our experiments to 3 epochs and shorter input sequences (128 tokens), our models still achieved strong performance on SST-2, QQP, and MRPC, validating LoRA's robustness with limited resources.

In the broader research area, LoRA is a foundational technique for parameter-efficient adaptation. Recent work such as ALoRA (Wang et al., 2024) introduced dynamic rank adaptation, where the rank r is adjusted during training based on layer-wise importance scores, further improving the efficiency of LoRA. RA-LoRA (Xu et al., 2024) introduced rank-adaptive low-rank adaptation for quantized language models, dynamically adjusting ranks during training to enhance efficiency and extend LoRA's principles to resource-constrained transformer architectures. Both of these papers demonstrate ongoing work being done with LoRA and its role in addressing the scalability changes of modern models. Our project also reinforces the practical value of LoRA in constrained settings.

5. Conclusions/Reflections

Key takeaways:

- Even at r=1, our fine-tuned models performed within $\sim 1\%$ of higher ranks ($r = 8, 64$). This minimal gap underscores LoRA's parameter efficiency and suggests that a fixed low rank often suffices.
- Adapting only W_q, W_v yielded almost identical accuracy to adapting all four projections W_q, W_v, W_k, W_o. In practice, this means you can save additional parameters by targeting just the most influential attention matrices without sacrificing performance.

Extensions and/or Future Directions:

- Migrate all experiments from CPU to scalable GPU cloud infrastructure. This would enable reproducing the original epoch counts and exploring larger sequence lengths.
- Implement adaptive rank schedules during fine-tuning similar to ALoRA to allocate more capacity to critical layers, maximizing accuracy per parameter.
- Evaluate LoRA beyond NLP—e.g., vision transformers, speech models, or multimodal tasks—to understand its effectiveness (or limitations) when handling dense inputs like images and/or audio spectrograms.

6. Tables and figures

6.1. Tables

Task	LoRA Accuracy (Paper)	Our Implementation Accuracy	Difference
SST-2	95.1±0.2	93.8±0.2	−1.3
QQP	90.8±0.1	88.7±0.2	−2.1
MRPC	89.7±0.7	88.0	−1.7

Table 1. Comparison of LoRA paper results and our implementation on GLUE tasks.

Hyperparameter	SST-2	QQP	MRPC
r	8	8	8
α	16	16	16
Epochs	3	3	3
Batch size	16	16	16
Learning rate	5e-4	5e-4	4e-4
Max seq length	128	128	128
Warm-up ratio	0.06	0.06	0.06
Target weights	Query + Value	Query + Value	Query + Value

Table 2. Hyperparameters used for training

Weight Type	r=1	r=2	r=4	r=8	r=64
W_q	72.74	77.50	79.57	81.50	83.23
W_q, W_v	85.03	85.45	86.42	86.53	87.28
W_q, W_v, W_k, W_o	85.50	85.89	86.54	86.86	87.29

Table 3. Validation accuracy (%) on MultiNLI for different rank r and weight types.

6.2. Figures

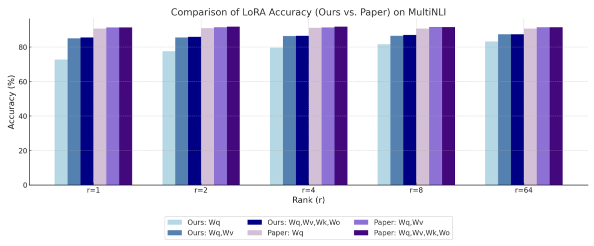

Figure 1. Visual representation of the results from Table 1.

References

[1] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). Lora: Low-rank adaptation of large language models. ICLR, 1(2), 3.

[2] Huggingface Datasets and Transformers libraries

[3] PyTorch Documentation: https://pytorch.org/docs/