

The LoRA(x)

Stanford CS224N Default Project

Ryan Da

Department of Computer Science
Stanford University
rda1@stanford.edu

Gerwin Mateo

Department of Computer Science
Stanford University
gerwin08@stanford.edu

Abstract

Large pretrained language models are typically adapted to downstream tasks through full fine-tuning, which updates all model parameters. While this can be effective, this approach can be computationally expensive and requires storing a separate full model copy for each task, which for smaller teams or even just our project, is a time-consuming thing. Parameter-efficient fine-tuning methods address this limitation by modifying only a small subset of model parameters while keeping the majority of the pretrained weights fixed, so that we only have to train on only a small subset of parameters. In this project, we investigate Low-Rank Adaptation (LoRA) [1] as an alternative to full fine-tuning for adapting GPT-2 [2] to two tasks from the default project, paraphrase detection and sonnet generation. We implemented and compared three training strategies, we first tried full fine-tuning of GPT-2 as basically a control, LoRA using the Hugging Face PEFT framework to see an already standard library, and a custom LoRA implementation that injects low-rank update matrices into attention layers to customize and see changes we make. Experiments are conducted on the Quora Question Pairs dataset for paraphrase detection and a Shakespeare sonnet dataset for text generation. We then evaluate the performance by using classification accuracy for paraphrase detection and chrF score [3] for sonnet generation.

Our results show that parameter-efficient fine-tuning can match or slightly outperform full fine-tuning while updating fewer than 1% of the model parameters. In particular, LoRA with higher rank configurations achieves the best performance for sonnet generation, surpassing the baseline full fine-tuning model. These findings demonstrate that LoRA provides an effective and efficient approach for adapting pretrained transformer models to new tasks without requiring full model retraining.

1 Key Information to include

- Mentor: Wei Liu
- External Collaborators (if you have any): None
- Sharing project: No

2 Introduction

Large-scale pretrained language models have become the dominant practice in modern natural language processing. Models such as GPT-2 [2] learn rich contextual representations from large corpus and can be adapted to downstream tasks through fine-tuning, which is demonstrated in our Default Project. In the standard fine-tuning setting, all model parameters are updated during training, which will then help with accuracy. While this approach often yields strong performance, it becomes increasingly expensive as model sizes grow into the hundreds of millions or billions of parameters.

Parameter-efficient fine-tuning methods attempt to address this limitation by modifying only a small subset of parameters while leaving the original pretrained weights unchanged. One such method is

Low-Rank Adaptation (LoRA) [1], which injects trainable low-rank matrices into existing weight layers of a transformer model. Instead of updating the original weight matrix directly, LoRA learns a low-rank decomposition that approximates the required update. This reduces the number of trainable parameters and memory requirements while maintaining comparable performance.

In this project, we explore the effectiveness of LoRA for adapting GPT-2 to two tasks included in the default project, paraphrase detection and sonnet generation. Paraphrase detection requires the model to determine whether two sentences express the same semantic meaning, while sonnet generation requires producing text that follows stylistic patterns similar to Shakespearean poetry.

We compare three training approaches, full fine-tuning of GPT-2, LoRA using the Hugging Face PEFT library, and a custom LoRA implementation integrated into the transformer attention layers. Then by evaluating these methods under comparable training conditions, we aim to understand how parameter-efficient fine-tuning affects performance, training stability, and model efficiency across both classification and generation tasks.

3 Related Work

Transformer-based language models have achieved strong performance across many natural language processing tasks. GPT-2 [2] showed that large autoregressive models trained on large-scale web text can perform a wide variety of tasks after being fine-tuned on specific datasets. Since then, pretrained language models have become the standard approach for many downstream NLP tasks.

Traditional fine-tuning updates all parameters of a pretrained model for a new task. While this approach is a normal approach and it often leads to good performance, it becomes more expensive as model sizes grow. Training and storing separate fully fine-tuned models for different tasks can require significant computational resources and memory. Because of this, recent research has focused on parameter-efficient fine-tuning methods that reduce the number of trainable parameters while still maintaining good performance.

Low-Rank Adaptation (LoRA) [1] is a widely used parameter-efficient fine-tuning method. Instead of directly updating the pretrained weights, LoRA freezes the original model parameters and adds small trainable low-rank matrices to certain weight layers during training. The authors show that many updates learned during fine-tuning are naturally low-rank, which means they can be approximated using smaller matrices while still achieving performance similar to full fine-tuning.

Recent work has integrated LoRA into large-scale training pipelines through libraries such as the Hugging Face PEFT framework. These tools make it easier to experiment with parameter-efficient training methods. They are especially useful when adapting large models to multiple tasks, since they reduce both storage requirements and computational costs compared to training separate fully fine-tuned models.

4 Motivation

While full fine-tuning of large pretrained language models often yields strong performance, it comes with several limitations. First, updating all model parameters requires significant computational resources and memory. For models with hundreds of millions or billions of parameters, this can make fine-tuning infeasible for small teams or projects with limited hardware. This is also a case in our project, where we don't have access to very performance heavy hardware. Second, storing fully fine-tuned copies of a model for multiple tasks quickly becomes costly, particularly when adapting to many downstream tasks.

Parameter-efficient methods like LoRA address these issues by updating only a small subset of parameters while keeping the majority of pretrained weights fixed. This allows for training multiple task-specific adapters without duplicating the full model, reducing both storage and compute requirements. Furthermore, recent research suggests that task-specific updates in pretrained transformers are often low-rank in nature, making it seem that methods like LoRA can capture the most important adaptations with a fraction of the parameters. So when making this, we had to think of would this adapt in real world settings as well.

In this project, our motivation is twofold. First, we aim to evaluate whether LoRA can maintain performance comparable to full fine-tuning. Second, we want to understand how LoRA’s hyperparameters, such as rank and scaling factor, affect model performance and training stability. This analysis helps inform best practices for adopting parameter-efficient fine-tuning in future projects.

5 Approach

Our approach compares three methods for adapting GPT-2 to downstream tasks. We will do this by doing full fine-tuning, LoRA using the PEFT framework, and a custom LoRA implementation. We evaluate on two tasks, paraphrase detection (binary classification from a prompt) and sonnet generation (next-token prediction over the full sequence). Our baseline is full fine-tuning of all GPT-2 parameters; we then compare against PEFT LoRA and our custom LoRA under the same task setups and evaluation metrics.

5.1 Low-Rank Adaptation (LoRA)

LoRA modifies transformer layers by decomposing weight updates into two low-rank matrices. For a pretrained weight matrix $W \in \mathbb{R}^{d \times k}$, LoRA introduces two matrices:

$$\Delta W = BA$$

where $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$ with rank $r \ll \min(d, k)$.

During training, the pretrained weights remain frozen and only the low-rank matrices are updated. This significantly reduces the number of trainable parameters while still allowing the model to adapt to new tasks. The forward pass for an adapted layer uses the scaling factor α/r , so the effective update is $\frac{\alpha}{r} \cdot BA$; we fix α and vary r in experiments. We apply LoRA only to attention projection layers, leaving the rest of the backbone frozen.

5.2 PEFT Implementation

We implement LoRA using the Hugging Face PEFT library, which integrates parameter-efficient fine-tuning into transformer models. LoRA modules are applied to attention projection layers including query, key, and value matrices. We use `LoraConfig` and `get_peft_model` to inject LoRA into our GPT-2 backbone as we did not implement the PEFT library itself. Target modules are the query, key, and value projections, and for sonnet generation we also include the attention output projection so that PEFT and our custom LoRA target the same layers for fair comparison.

5.3 Custom LoRA Implementation

To better understand the mechanism of LoRA, we also implement our own LoRA modules directly within the transformer architecture. This involves inserting low-rank matrices into selected linear layers and ensuring that only these parameters are updated during training. This custom implementation allows us to experiment with different ranks, learning rates, and target modules. Our wrapper computes output $= xW^\top + (\alpha/r)xA^\top B^\top$, with W frozen and only A, B trainable. We initialize A with Kaiming uniform and B with zeros so the low-rank term is zero at initialization. This implementation is our own code and allows us to vary rank, α , learning rate, and target modules in a controlled way.

6 Experiments

6.1 Data

We evaluate our models on two tasks from the default project.

For paraphrase detection, we use the Quora Question Pairs dataset, which contains pairs of questions labeled as paraphrases or non-paraphrases. The dataset is commonly used to evaluate semantic similarity models.

For sonnet generation, we use a dataset of Shakespearean sonnets. During training, the model learns to generate the continuation of a sonnet given the first few lines as input. This task tests the model’s ability to learn stylistic and structural patterns of poetry.

6.2 Evaluation method

For paraphrase detection, we evaluate model performance using classification accuracy on the development and test sets.

For sonnet generation, we use the chrF metric [3], which measures similarity between generated text and reference text using character-level n-grams. This metric is commonly used in text generation tasks and provides a robust measure of textual similarity.

6.3 Experimental details

All experiments use the GPT-2 small configuration with approximately 124 million parameters. Models are initialized with pretrained weights from Hugging Face.

Baseline models are trained using full fine-tuning with a learning rate of 1×10^{-5} . LoRA models use higher learning rates since only a small subset of parameters are updated.

For LoRA experiments, we vary the rank parameter and the set of target modules in the transformer architecture. Higher-rank configurations allow the model to learn more expressive updates while still maintaining parameter efficiency.

Training is conducted for multiple epochs, and model selection is based on performance on the development set.

6.4 Results

We evaluate the performance of full fine-tuning and parameter-efficient fine-tuning methods on both tasks in our project, sonnet generation and paraphrase detection. For sonnet generation, performance is measured using the chrF metric, which evaluates character-level similarity between generated and reference text. For paraphrase detection, we measure classification accuracy on the development set.

Sonnet Generation Table 1 shows the results for sonnet generation across different training strategies. The baseline full fine-tuning model achieves a development chrF score of 40.04. Early LoRA experiments using rank 8 achieve a slightly higher peak score of 40.84 but exhibit instability during later epochs, with scores fluctuating between 34 and 37. The PEFT implementation with the same rank underperforms the baseline, achieving a maximum chrF score of 37.20 and showing little improvement during training.

To improve training stability and model capacity, we increase the LoRA rank to 16 and tune several hyperparameters including the scaling factor ($\alpha = 32$), learning rate (3×10^{-4}), and the set of target modules within the attention layers. These changes significantly improve performance for both LoRA implementations. The custom LoRA implementation with rank 16 achieves the a best overall result with a chrF score of **41.429**, while the PEFT implementation reaches a comparable score of 41.41. Both outperform the baseline full fine-tuning model and also a little bit of the staff baseline.

Model	Best Dev chrF
Full Fine-Tuning (Baseline)	40.04
LoRA (rank 8)	40.84
PEFT LoRA (rank 8)	37.20
LoRA (rank 16)	41.429
PEFT LoRA (rank 16)	41.41

Table 1: Sonnet generation performance measured using chrF on the development set.

Paraphrase Detection We next evaluate model performance on the paraphrase detection task using the Quora Question Pairs dataset. Table 3 reports the development set accuracy for both full fine-tuning and LoRA-based training.

	Baseline	LoRA r8	PEFT r8	LoRA r16	PEFT r16
Rank (r)	–	8	8	16	16
α	–	16	16	32	32
Learning Rate	$1e^{-5}$	$1e^{-4}$	$1e^{-4}$	$3e^{-4}$	$3e^{-4}$
Final Dev Loss	3.12	~ 3.2	4.12	2.88	3.37
Best Epoch(out of 30)	26	early (10–12)	25	27	28

Table 2: Training configurations and final development loss for the different models.

The baseline full fine-tuning model achieves a development accuracy of 0.893. The LoRA-based model with rank 16 achieves a development accuracy of 0.887, which is only slightly lower than the baseline despite updating significantly fewer parameters. In our implementation, LoRA trains approximately 0.7% of the total model parameters while keeping the remaining weights frozen.

These results demonstrate that parameter-efficient fine-tuning can maintain performance comparable to full fine-tuning while dramatically reducing the number of trainable parameters.

Model	Dev Accuracy
Full Fine-Tuning (Baseline)	0.893
LoRA (rank 16)	0.887
PEFT LoRA (rank 16)	0.883

Table 3: Paraphrase detection accuracy on the Quora development set.

Overall, our experiments show that LoRA can match or exceed full fine-tuning performance for generation tasks while maintaining strong performance for classification tasks. Increasing the rank of the low-rank adaptation matrices improves both training stability and final model performance, highlighting the importance of carefully tuning LoRA hyperparameters.

7 Analysis

Our results highlight several trade-offs between full fine-tuning and parameter-efficient methods.

First, LoRA significantly reduces the number of trainable parameters. In our experiments, fewer than one percent of the model parameters were updated during training when using LoRA. This reduces memory usage and makes it easier to train multiple task-specific adapters. This made it easier for us to run, specifically sonnet generation, without the need of an external GPU.

Second, training stability depends on hyperparameter choices such as the LoRA rank and learning rate. Lower-rank configurations sometimes showed unstable training behavior, whereas higher-rank configurations produced more consistent improvements. We also took some inspiration from John Schulman’s lecture on LoRA, and how we should explore some different learning rates and rank configurations. Beforehand, we weren’t getting to baseline with LoRA, but after taking into consideration changing both in conjunction, it helped a lot.

8 Conclusion

In this project, we explored parameter-efficient fine-tuning using Low-Rank Adaptation (LoRA) to adapt GPT-2 to two tasks from the default project. We compared three approaches, full fine-tuning of all model parameters, LoRA using the Hugging Face PEFT library, and our own custom LoRA implementation integrated directly into the transformer attention layers.

Our results show that LoRA can perform similarly to full fine-tuning while training far fewer parameters. For paraphrase detection, the LoRA model achieved accuracy close to the fully fine-tuned baseline even though only a small fraction of the parameters were updated. For sonnet generation, LoRA slightly outperformed the baseline model in terms of chrF score when using higher-rank configurations.

We also observed that the choice of hyperparameters plays an important role in LoRA performance. In particular, increasing the rank of the low-rank matrices improved training stability and final performance. Adjusting the learning rate and scaling factor also helped the model converge more reliably. These results suggest that while LoRA is efficient, it still requires careful tuning to achieve strong results.

Overall, our experiments demonstrate that parameter-efficient methods such as LoRA can be a practical alternative to full fine-tuning. By updating fewer than one percent of the model parameters, LoRA reduces computational cost and memory requirements while maintaining strong performance. This makes it especially useful when adapting large pretrained models to multiple tasks or when computational resources are limited.

In the future, we'd like to consider other ways to help with these tasks. Although LoRA is a great step in the right direction, we know as a team, we could have considered other approaches that would help with the process. For example, we could have updated the generate function on our sonnet to make sure we're capturing the right tokens.

Team contributions

Gerwin handled sonnet generation and the baseline, and wrote the custom LoRA code. Ryan handled paraphrase detection and found the references we cite. We both worked on LoRA and PEFT and tuned the models together.

References

- [1] Edward J. Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [2] Alec Radford, Jeffrey Wu, Rewon Child, et al. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.
- [3] Maja Popović. chrF: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, 2015.