

Improving GPT-2 Fine-Tuning through Parameter-Efficient Adaptation and Preconditioned Optimization

Stanford CS224N Default Project

Akhilesh Balasingam
Department of Computer Science
Stanford University
avb03@stanford.edu

Georgios Mikos
Department of Computer Science
Stanford University
mikosg@stanford.edu

Abstract

Generalist large language models such as GPT-2 present immense promise for a wide array of downstream tasks; however, adapting these models efficiently remains challenging due to their large parameter count and training cost. Efficient fine-tuning and optimization techniques may provide an avenue to reduce this training cost. After implementing core components of the GPT-2 architecture, we assess parameter-efficient fine-tuning methods (LoRA, DoRA, QLoRA, and ReFT) and second-order optimization techniques (Shampoo and SOAP) and examine their capacity to reduce the number of trainable parameters and accelerate training while maintaining comparable performance to full fine-tuning. Our results demonstrate DoRA achieves near full fine-tuning performance with 98% fewer trainable parameters and Shampoo achieves convergence with 50% fewer iterations, empirically demonstrating the potential for these methods to efficiently enable the training of inexpensive, domain-tailed models.

1 Key Information to include

- Mentor: Alisa Levin
- External Collaborators: No
- Sharing project: No
- Late Days:

Team member	Late days remaining
Akhilesh Balasingam	5
Georgios Mikos	5
Total pooled	10
Team size	2
Late days applied for the report	3

2 Introduction

Large pretrained language models such as GPT-2 achieve strong performance across many natural language processing tasks [1]. These models are typically adapted to downstream tasks through fine-tuning, where pretrained parameters are updated using task-specific data. However, full fine-tuning introduces several challenges. First, updating all parameters of a large model requires substantial memory and compute resources. Second, when training datasets are small, updating the entire model can lead to overfitting and unstable optimization.

Recent work has proposed several strategies to address these issues. Parameter-efficient fine-tuning (PEFT) methods aim to reduce the number of trainable parameters by introducing small trainable modules while keeping most pretrained parameters frozen. One widely used approach is LoRA (Low-Rank Adaptation), which injects low-rank updates into transformer weight matrices. Instead of directly modifying the full weight matrix, LoRA learns a low-rank decomposition of the update, allowing the model to adapt to new tasks while updating only a small fraction of parameters. Extensions such as DoRA and QLoRA further explore how low-rank updates and quantization can improve parameter efficiency while maintaining strong performance [2].

In addition to reducing parameter counts, improving the optimization process itself can significantly affect fine-tuning performance. Most large language models are trained with first-order optimizers such as Adam, which rely only on gradient information and discard second-order statistics. Second-order optimization methods incorporate curvature information to better precondition gradient updates. One such method is Shampoo, which approximates second-order curvature statistics to adaptively scale gradients, potentially improving convergence speed and stability compared to standard optimizers [3]. Another method is SOAP, which extends Shampoo by using Adam-like adaptive moment methods in the parameter eigenspace [4].

In this project, we investigate how parameter-efficient adaptation methods and improved optimization strategies affect the fine-tuning of GPT-2. In particular, we study:

- Parameter-efficient adaptation methods (LoRA, DoRA, QLoRA, ReFT)
- Second-order preconditioned optimization (Shampoo, SOAP)

We evaluate these approaches on sonnet generation and paraphrase detection tasks, and analyze the trade-offs between parameter efficiency, optimization behavior, and downstream model performance.

3 Related Work

3.1 Parameter-Efficient Fine-Tuning

Parameter-efficient fine-tuning methods aim to reduce the number of trainable parameters required to adapt large pretrained models. LoRA introduces low-rank matrices that parameterize updates to existing weight matrices while keeping the original weights frozen. This approach significantly reduces memory consumption and training cost [5].

Extensions to LoRA include DoRA, which decomposes weight updates into magnitude and directional components, improving the expressiveness of low-rank updates [6], and QLoRA, which combines LoRA with quantized base models to enable fine-tuning under strict memory constraints [7].

3.2 Second-Order Optimization

Most neural networks are trained using first-order methods such as stochastic gradient descent or Adam. Second-order optimization methods attempt to improve convergence by incorporating curvature information. Shampoo approximates second-order statistics using Kronecker-factored matrix preconditioners, enabling more efficient gradient updates [3]. Shampoo with Adam in the Preconditioner’s eigenbasis (SOAP) extends Shampoo by recognizing that it is equivalent to running Adafactor in the eigenbasis of the algorithm’s preconditioner and thus can be adapted to running Adam in that same basis [4]. These methods have recently been used in training large-scale neural networks such as Gemini 1.5 and offer potential benefits when fine-tuning large language models [8].

4 Approach

Our goal is to improve both the efficiency and stability of fine-tuning GPT-2. We explore two complementary modifications to the standard fine-tuning pipeline: parameter-efficient adaptation and second-order preconditioned optimization.

4.1 Standard Fine-Tuning Objective

Let $f(x; \theta)$ denote the pretrained GPT-2 model with parameters θ . For a dataset $\{(x_i, y_i)\}_{i=1}^n$, standard fine-tuning minimizes the empirical risk

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(f(x_i; \theta), y_i)$$

where $\ell(\cdot)$ is the task-specific loss function.

Training is typically performed using stochastic gradient updates

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} L(\theta_t)$$

However, this approach updates all parameters and may lead to inefficient training and unstable optimization.

4.2 Parameter-Efficient Adaptation

Parameter-efficient fine-tuning methods reduce the number of trainable parameters by introducing structured updates to pretrained models while keeping most weights frozen.

LoRA. For a weight matrix $W \in \mathbb{R}^{d \times k}$ in the transformer, LoRA parameterizes the update as

$$W' = W + BA$$

where $A \in \mathbb{R}^{r \times k}$, $B \in \mathbb{R}^{d \times r}$, and $r \ll \min(d, k)$. During training, the pretrained weights W are frozen and only the low-rank matrices A and B are updated. We implemented LoRA from scratch.

DoRA. DoRA extends LoRA by decomposing each weight vector into magnitude and direction components. For a weight vector w_j , we write $w_j = m_j v_j$, where $m_j = \|w_j\|_2$ is the magnitude and v_j is the unit direction vector. Instead of applying a low-rank update to the full weight matrix, DoRA updates the directional component using $V' = V + BA$, where $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$. The adapted weight matrix is then $W' = MV'$, where M is the diagonal matrix of pretrained magnitudes. This formulation preserves the original magnitude structure while allowing low-rank updates to modify the weight directions. We used the Hugging Face PEFT implementation [9].

QLoRA. QLoRA extends LoRA by storing pretrained weights in quantized form while maintaining LoRA adapters in higher precision. This reduces memory needs while preserving training stability. We used the bitsandbytes implementation [7].

ReFT (Representation Fine-Tuning). Unlike the previous approaches, which modify model weights, ReFT adapts the model by intervening directly in hidden representations. Let $h \in \mathbb{R}^d$ denote the hidden state at a given transformer layer. ReFT applies a learned low-dimensional intervention of the form $h' = h + Pz$, where $P \in \mathbb{R}^{d \times r}$ and $z \in \mathbb{R}^r$ with $r \ll d$. The vector z is produced by a small learned function of the representation, typically parameterized as $z = f_{\theta}(h)$ with a lightweight module (e.g., a small linear or low-rank layer). During training, the pretrained model weights remain frozen while only the intervention parameters θ and P are updated.

This approach adapts the model by modifying a small subspace of the representation space rather than altering the underlying weight matrices, allowing ReFT to achieve competitive performance while training significantly fewer parameters. We used the Stanford NLP pyreft implementation [10].

4.3 Second-Order Preconditioned Optimization

Finally, we explore second-order optimization methods based on Shampoo. Instead of performing standard gradient descent updates, in the naive implementation, Shampoo preconditions the gradient using approximate curvature matrices.

Let $G_t = \nabla_{\theta} L(\theta_t)$. The update rule becomes

$$\theta_{t+1} = \theta_t - \eta L_t^{-1/4} G_t R_t^{-1/4}$$

where $L_t = L_{t-1} + G_t G_t^T$ and $R_t = R_{t-1} + G_t^T G_t$, thus creating a preconditioning matrix constructed from second-order statistics of past gradients. By adapting the geometry of the update space, Shampoo can improve convergence speed and training stability. SOAP performs a similar update with with Adam-like updates in the rotated eigenspace:

$$\theta_{t+1} = \theta_t - \eta Q_L N'_t Q_R^T$$

where the Adam-like update entails $N'_t = M'_t / \sqrt{V_t}$, with $V_t = \beta_2 V_{t-1} + (1 - \beta_2)(G'_t \odot G'_t)$ and preconditioning $G'_t = Q_L^T G_t Q_R$, $M_t = \beta_1 M_{t-1} + (1 - \beta_1)G_t$, $M'_t = Q_L^T M_t Q_R$, $L_t = \beta_2 L_{t-1} + (1 - \beta_2)G_t G_t^T$, $R_t = \beta_2 R_{t-1} + (1 - \beta_2)G_t^T G_t$, $Q_L = \text{Eigenvectors}(L)$, and $Q_R = \text{Eigenvectors}(R)$. The preconditioning frequency controls how often Q_L and Q_R are updated. Although we originally implemented Shampoo from scratch, we turned to the Meta Research optimizers implementation owing to their better numerical methods for fractional matrix powers [11].

5 Experiments

We evaluate these fine-tuning methods on multiple downstream tasks, covering both classification and generation settings. These experiments allow us to study the effects of parameter-efficient adaptation and optimization techniques on model performance, training stability, and memory efficiency.

5.1 Data

Stanford Sentiment Treebank (SST). SST consists of movie review sentences labeled with five sentiment categories ranging from negative to positive. We use the provided splits of 8,544 training examples, 1,101 development examples, and 2,210 test examples. The task is formulated as sentiment classification using the final token representation from GPT-2.

CFIMDB. The CFIMDB dataset contains longer movie reviews labeled with binary sentiment (positive or negative). It includes 1,701 training examples, 245 development examples, and 488 test examples. This dataset evaluates model performance on longer text sequences.

Quora Question Pairs. For paraphrase detection we use a subset of the Quora Question Pairs dataset containing 141,506 training examples, 20,215 development examples, and 40,431 test examples. Following the project setup, we formulate this task as a close-style generation problem where GPT-2 predicts the token “yes” or “no” indicating whether the two questions are paraphrases.

Shakespeare Sonnet Dataset. The sonnet dataset contains 154 Shakespearean sonnets, each consisting of 14 lines. We use 143 sonnets for training and 12 for testing. During evaluation, the model is conditioned on the first three lines of a sonnet and tasked with generating the remaining lines.

5.2 Evaluation Method

For the sentiment analysis and paraphrase detection tasks we report classification accuracy on the development and test sets. For the sonnet generation task we evaluate generated poems using the chrF metric, which measures character-level similarity between generated and reference sonnets.

In addition to task performance, we analyze training efficiency by reporting the number of trainable parameters, training loss, training time, throughput (tokens/sec), and peak memory usage.

5.3 Experimental Details

All experiments use the GPT-2 small architecture as the base model. For parameter-efficient fine-tuning we evaluate LoRA, DoRA, QLoRA, and ReFT adapters on both attention and MLP layers.

For LoRA-based methods we sweep the adapter rank $r \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$, learning rates $\{5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4}\}$, and batch sizes $\{4, 8\}$. These configurations vary the fraction of trainable parameters from approximately 0.1% to 38% of the full model. For ReFT,

we instead vary the dimensionality of the learned representation intervention using the same set of ranks r , along with the same learning rate and batch size sweeps. Because ReFT modifies hidden representations rather than weight matrices, the number of trainable parameters is substantially smaller, ranging from approximately 0.001% to 0.1% of the full model.

For second-order optimization methods Shampoo and SOAP, we perform initial hyperparameter sweeps at a 10% data down sample, owing to the computation cost of each experiment. We sweep learning rates $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ and preconditioner frequency updates $\{10, 100, 1000\}$. After identifying the optimal method and hyperparameter, we return to the full datasets.

5.4 Results

We evaluate the effectiveness of parameter-efficient fine-tuning (PEFT) methods across a range of adapter ranks and hyperparameter configurations. Our analysis compares LoRA, DoRA, QLoRA, and ReFT in terms of model performance, parameter efficiency, and computational cost.

As an initial experiment, we performed a LoRA hyperparameter sweep on the Quora paraphrase detection task and achieved a maximum accuracy of 0.882. However, due to our limited compute budget, it was not feasible to conduct the same exhaustive sweeps across all tasks. We therefore selected a single task for deeper analysis. Parameter-efficient methods are particularly attractive in settings with limited data or compute, where full fine-tuning can be both expensive and prone to overfitting. Motivated by this, we focus the remainder of our analysis on the Shakespeare sonnet generation task, which allows us to systematically study how PEFT methods affect generation quality, parameter efficiency, and training dynamics.

Conversely, second-order optimization techniques excel in high-data, low compute availability regimes because by incorporating curvature statistics they may be able to reach optimal model convergence faster, ultimately requiring fewer passes through the entire dataset. Motivated by this rational and similar constrained by our compute budget, we focused our analysis of second-order optimization techniques on the Quora paraphrase detection task.

5.4.1 Parameter-Efficient Fine-tuning

Figure 1 summarizes the main experimental findings. Performance improves rapidly as the adapter rank increases (Fig. 1a), but gains saturate beyond moderate ranks. At the same time, strong performance can be achieved with only a small fraction of trainable parameters (Fig. 1b), demonstrating the effectiveness of parameter-efficient adaptation. Comparing methods across all hyperparameter configurations shows that DoRA achieves the highest median performance while QLoRA exhibits slightly higher variance (Fig. 1c). Memory usage increases approximately linearly with rank (Fig. 1d), highlighting the trade-off between performance and resource cost. Across the full hyperparameter sweep, the best configuration occurs at rank $r = 32$ with batch size 4 and learning rates between 1.5×10^{-4} and 2×10^{-4} . In particular, DoRA achieves the highest chrF score of 40.47, while LoRA and QLoRA reach 39.88 and 40.08, respectively. ReFT achieves 40.09 while using only 0.11% of the model parameters as trainable weights (Fig. 1e), illustrating that representation-level interventions can match the performance of adapter-based methods with substantially smaller trainable fractions. On the test set, DoRA achieved a chrF score of 40.294, whereas full fine-tuning achieved 41.828.

5.4.2 Second Order Optimizers

Figure 2 summarizes the main experimental findings. Performance of the Adam, Shampoo, and SOAP optimizers are comparable across four orders of learning rates, with Shampoo demonstrating the highest performance on the sub-sample of data used for this hyperparameter sweep, thus validating the usefulness of second-order methods (Fig. 2a). Interestingly, the second-order awareness in Shampoo permits near-ideal performance with a higher learning rate that allows for faster model convergence. Modifying the precondition frequency that controls the frequency of updates to the second order statistics, a key computational bottleneck of second-order methods, unexpectedly demonstrates ideal performance with 100 steps per update for Shampoo and 1000 steps per update for SOAP, possibly because more frequent updates result in moderate instability (Fig. 2b). Comparing Adam and Shampoo with optimal hyperparameters on the full dataset, Adam achieves a dev accuracy of 0.895 and test accuracy of 0.863, whereas Shampoo achieves a dev accuracy of 0.892 and test accuracy of 0.858, even though Shampoo achieves a lower training loss with fewer iterations.

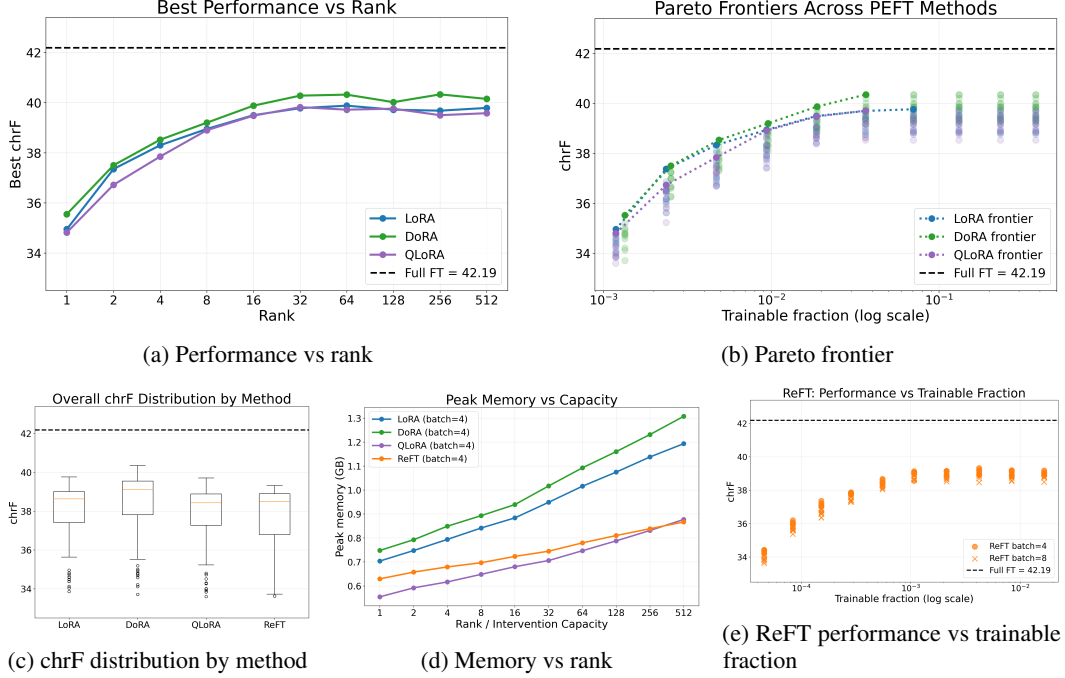


Figure 1: Comparison of parameter-efficient fine-tuning methods. (a) Performance improves with increasing adapter capacity before saturating. (b) Pareto frontiers show the tradeoff between parameter efficiency and downstream performance. (c) The overall chrF distributions indicate that DoRA tends to achieve the strongest performance among the PEFT methods considered, while QLoRA and ReFT remain competitive. (d) Memory usage differs substantially across methods, with QLoRA offering the lowest footprint and ReFT also remaining relatively lightweight. (e) ReFT operates at a much smaller trainable fraction than the other methods while still achieving competitive performance.

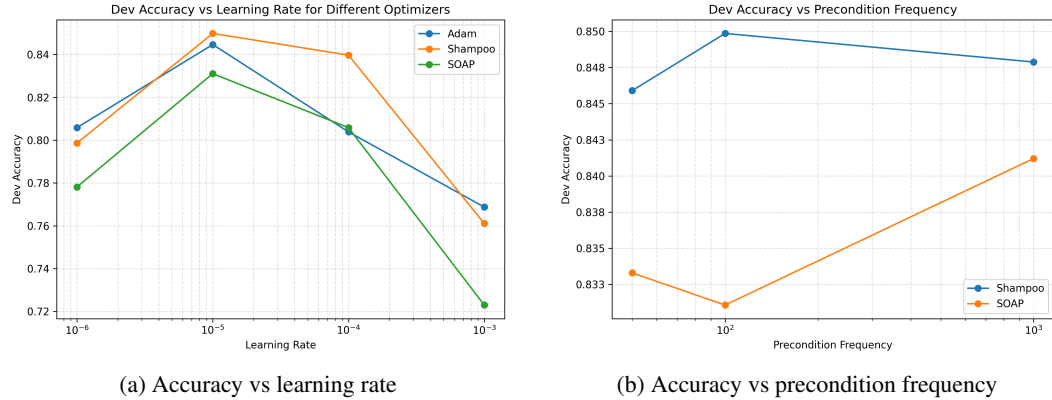
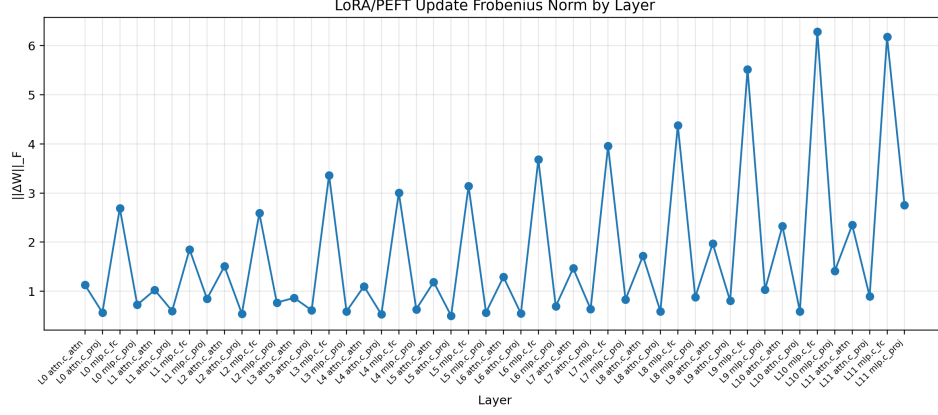


Figure 2: Comparison of optimization methods. (a) All models share an optimal learning rate, with Shampoo achieving the best performance. (b) Modifications of the precondition frequency impacts final accuracy non-uniformly across methods.

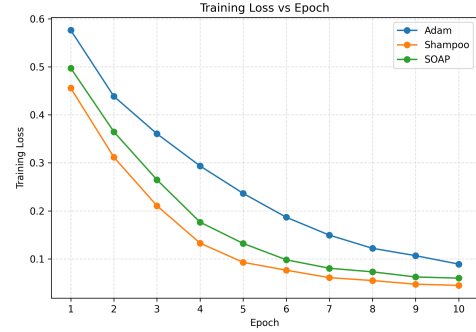
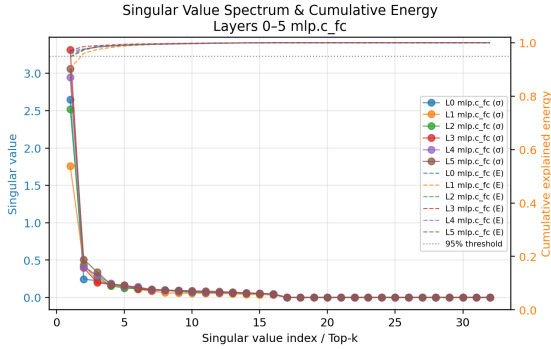
6 Analysis

6.1 Parameter-Efficient Fine-tuning

Update Magnitude Across Layers Figure 3a shows the Frobenius norm of the learned LoRA update matrix ΔW for each adapted layer. The Frobenius norm is defined as



(a) Frobenius norm of the LoRA update $\|\Delta W\|_F$ across layers.



(b) Singular value spectrum and cumulative explained energy of LoRA updates.

(c) Shampoo and SOAP demonstrate improved convergence.

Figure 3: Analysis of LoRA updates across GPT-2 and training loss with second-order optimizers.

$$\|\Delta W\|_F = \sqrt{\sum_{i,j} (\Delta W_{ij})^2},$$

which measures the overall magnitude of the weight update introduced by LoRA. Because the base pretrained weights W remain frozen during parameter-efficient fine-tuning, ΔW captures the entire task-specific adaptation. Larger norms therefore indicate layers where the model performs stronger adjustments to the pretrained representation.

Interestingly, the magnitude of updates varies significantly across layers, suggesting that adaptation is not uniformly distributed throughout the network. Certain feedforward layers exhibit larger updates, indicating that the downstream task primarily modifies specific components of the transformer rather than requiring global reconfiguration of the entire model.

Low-Dimensional Structure of LoRA Updates To further analyze the structure of the learned updates, we examine the singular value decomposition (SVD) of each update matrix:

$$\Delta W = U \Sigma V^T,$$

where $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r)$ contains the singular values. Each singular value represents the strength of a principal direction in which the update modifies the weight matrix.

Because LoRA parameterizes updates as $\Delta W = BA$, with $A \in \mathbb{R}^{r \times d}$ and $B \in \mathbb{R}^{d \times r}$, the rank of ΔW is theoretically bounded by r . However, the effective rank of the update can be significantly smaller if most of the singular value mass is concentrated in only a few components.

Explained Energy of Singular Values To quantify this phenomenon, we compute the cumulative explained energy of the singular values:

$$E_k = \frac{\sum_{i=1}^k \sigma_i^2}{\sum_{i=1}^r \sigma_i^2}.$$

This metric measures the fraction of the Frobenius norm captured by the top- k singular directions. Because $\|\Delta W\|_F^2 = \sum_{i=1}^r \sigma_i^2$ the cumulative energy curve indicates how quickly the update matrix becomes well approximated by a low-rank representation.

As shown in Figure 3b, the explained energy rises very quickly: only a small number of singular values account for nearly all of the update magnitude. This implies that the learned updates occupy a very low-dimensional subspace of the full weight matrix space.

Interpretation These observations provide empirical evidence for the central hypothesis behind parameter-efficient fine-tuning. Although modern transformer models contain hundreds of millions of parameters, the directions in weight space required to adapt the model to a new task appear to lie in a much lower-dimensional manifold.

Rather than requiring arbitrary modifications to the full parameter matrix, the optimization process consistently concentrates the update into a few dominant directions, particularly at the later layers in the model. This suggests that much of the pretrained model’s structure already provides a strong inductive bias for the task, and fine-tuning primarily needs to adjust a small set of task-specific transformations. From this perspective, LoRA does not merely reduce the number of trainable parameters; it also implicitly exploits a geometric property of the fine-tuning problem, namely that useful adaptations occur within a low-dimensional subspace of the original parameter space.

6.2 Second Order Optimization

Learning Rates and Convergence To assess whether incorporating second-order information accelerates the model’s convergence, we examine the training loss, an indicator of how well the optimizer manages to fit the training data. At a learning rate of 10^{-4} , the learning rate that produced the lowest training losses, Shampoo and SOAP demonstrate significantly faster convergence, reaching a loss value below 0.1 with 50% and 40% fewer iterations, respectively, as seen in Figure 3c.

Interpretation Second-order optimization methods demonstrate significantly faster model convergence, as observed by the training loss curves. However, this improved convergence does not fully translate to improved accuracy on the held-out dataset, illustrating the difference between reaching a local minimum in the training loss space that almost certainly represents an over-fit model versus finding a set of parameters along the training path that perform well on the held-out data.

7 Conclusion

Here, we analyze strategies for efficient fine-tuning of GPT-2, namely parameter-efficient fine-tuning strategies and second-order optimization techniques. We find that DoRA performs best on the Sonnet-generation task, achieving near full fine-tuning performance with only 2% as many parameters, while Shampoo achieves 50% faster convergence and comparable performance to Adam on the held-out data. Together, these results provide compute and memory efficient strategies for fine-tuning that could accelerate the adoption of domain-specific models, and future work should examine whether these conclusions apply with larger, more capable models and the synergistic effects of PEFT and optimization strategies. Limitations of this study include the low task diversity and omission of an analysis of regularization techniques that alter first and second order optimizer performance.

Team contributions

AB and GM conceptualized and designed this study, AB performed all PEFT experiments, GM performed all optimization experiments, and AB and GM wrote the report.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners.
- [2] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. (arXiv:2403.14608).
- [3] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. (arXiv:1802.09568).
- [4] Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and stabilizing shampoo using adam. (arXiv:2409.11321).
- [5] 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. (arXiv:2106.09685).
- [6] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. (arXiv:2402.09353).
- [7] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. (arXiv:2305.14314).
- [8] Google Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context.
- [9] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, Benjamin Bossan, and Marian Tietz. PEFT: State-of-the-art parameter-efficient fine-tuning methods. <https://github.com/huggingface/peft>, 2022.
- [10] Zhengxuan Wu, Atticus Geiger, Aryaman Arora, Jing Huang, Zheng Wang, Noah D. Goodman, Christopher D. Manning, and Christopher Potts. pyvene: A library for understanding and improving PyTorch models via interventions. 2024.
- [11] Hao-Jun Michael Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat. A distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale. https://github.com/facebookresearch/optimizers/tree/main/distributed_shampoo, 2023.