

LoRA – Low Rank Adaptation of Large Language Models

(https://github.com/kumara3/DeepDive_LORA)

Ashwani Kumar (axk200017), Alexander Ntafos (acn180003)
Nikhil Hegde(nxh230013), Paul Mason(pjm210000)

Abstract

Large language models (LLMs) need to be optimized for specific purposes, such as spam categorization, in order to attain high accuracy. However, typical comprehensive fine-tuning requires updating billions of parameters, resulting in prohibitively high memory and computing costs. Because of this difficulty, deployment in situations with limited resources is not feasible. We investigate Low-Rank Adaptation (LoRA), a parameter-efficient fine-tuning method that incorporates small trainable low-rank matrices into Transformer layers and freezes pre-trained weights. LoRA maintains competitive performance while drastically reducing the number of trainable parameters. In this study, we use GPT-2 to apply LoRA to the SMS Spam Collection dataset and assess its efficacy across various rankings and scaling parameters. The results show that LoRA achieves almost perfect fine-tuning accuracy with up to $10\times$ less trainable parameters, reducing GPU memory consumption and training time. This technique enables the scalable and cost-effective adaptation of LLMs for real-world applications such as edge deployment, spam detection, and personalization.

1 Introduction

Before we can properly use large pre-trained language models like LLMs for tasks in downstream natural language processing like text classification, sentiment analysis, or spam detection, they need some task-specific tweaking first. Models such as BERT and GPT are trained on a large amount of diverse data, but they might still need specific adjustments to really hit their peak performance. This is because their general knowledge does not always transfer optimally to their new task in a different domain. This adjustment phase, often called full fine-tuning, typically involves updating all the parameters in the model. While this can lead to great accuracy and performance, it does demand a good deal of memory and processing power.

Fine-tuning large models is no small feat, and there are many reasons why it takes so many resources. To start with, processing and updating billions of parameters requires a significant amount of GPU or TPU resources, which leads to long training times and high

energy usage. Then there's the problem that these models require a lot of memory to function, making them impractical in many real world or resource limited situations. Additionally, having to maintain separate optimized copies of models for various tasks introduces multiple scalability and storage issues. For example, GPT-3 has 175 billion parameters, so storing multiple full fine-tuning copies of it for multiple applications and tasks becomes excessively and prohibitively expensive.

These major problems have resulted in the need for and development of parameter-efficient fine-tuning methods. These methods focus on adapting large models by only updating a small subset of the weights. Previous attempts to fix this problem like adapter layers, prompt tuning, prefix tuning, and bias only updates helped to reduce the processing and storage demands, but each one of them came with their own tradeoffs. Like problems with training stabilities, expressiveness, or latency.

Low-Rank Adaptation, or LoRA for short, is a practical solution to these limitations and problems. Instead of updating the full weight matrices, it fine-tunes models efficiently by adding the update as small, trainable low-rank matrices to the Transformer setup while keeping the original pre-trained weights intact (frozen). By focusing weight updates on just a low-rank decomposition, it greatly reduces the number of parameters that need training, all without compromising performance or increasing latency. This approach is especially useful in real-world scenarios where computing power is limited, as it allows for faster training, uses less memory, and makes it easier to deploy models across different tasks.

In this project, we will show the motivation behind Low-Rank Adaptation (LoRA), explain how it works, compare it to prior approaches, and evaluate its performance in our own practice example.

2 Related work

Transformer-based architectures have changed natural language processing (NLP) by using self-attention methods. These approaches help models understand long-range dependencies and context in text. Two key models in this group are BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer). BERT's ability to understand context from both directions makes it great for tasks such as text classification and question answering. On the other hand, GPT's autoregressive method makes it better for generating text and for conversational AI.

To adjust these large pre-trained models for specific tasks, extensive fine-tuning is usually necessary. In this process, all model parameters are updated using data tailored to the task. Although this method often results in high accuracy, it has some important downsides:

Computational Cost: Long training times and significant GPU or TPU resources are needed to update billions of parameters.

Memory Footprint: Large models need a lot of VRAM, making them unsuitable for environments with limited resources.

Scalability Issues: Storing multiple refined versions for different purposes leads to high storage demands and deployment challenges.

Researchers have developed Parameter-Efficient Fine-Tuning (PEFT) methods to solve these problems. These techniques focus on keeping performance steady while lowering the number of trainable parameters. Key PEFT techniques include:

Prompt tuning: This method guides the model without changing all weights by introducing task-specific prompts, which can be hard or soft.

Prefix tuning: It increases flexibility by adding trainable continuous embeddings, or soft prompts, to the input sequence.

Adapters: These reduce the need to change the whole network by inserting small trainable modules between Transformer layers.

Low-Rank Adaptation (LoRA): This greatly reduces trainable parameters while preserving inference performance by reparametrizing weight updates into low-rank matrices.

Among these, LoRA stands out for its simplicity and effectiveness. It improves fine-tuning by adding low-rank matrices to attention layers while keeping the original model weights fixed. This method is efficient and requires less computing power. As a result, it is particularly suitable for large-scale language models and real-world applications where cost and scalability are important factors.

3 Methodology

3.1 Problem Statement

The findings from previous studies [1,2,3,4] have shown that the learned over-parameterized models reside on a low intrinsic dimension. Further, weight matrix changes during task-specific adaptation take place in a subspace that is of significantly lower rank than the initial parameter space. Hence, low-rank decomposition can efficiently describe the fine-tuning update matrix, represented by ΔW , without compromising performance. This forms the foundational basis of working of LoRA. The low-rank aspect refers to the mathematical concept of limiting model adjustments to a smaller dimensional subspace of the total weight parameter space. This effectively captures the most influential directions of the weight parameter changes during training.

3.2 Motivation

Let $P\Phi(y|x)$ be a large multi-task model such as GPT based on the Transformer architecture [5]. We are interested in adapting this model for downstream tasks such as summarization, classification, or machine reading comprehension. Let each downstream task be defined by a dataset of context-target pairs as shown below: -

$$Z = (x_i, y_i)_{i=1}^N; \text{ where } x_i, y_i \text{ are sequences of tokens.}$$

In the standard full fine-tuning, the model parameters are initialized at the pre-trained weights Φ_0 and updated to $\Phi_0 + \Delta\Phi$ by maximizing the conditional language modeling objective function [1] as below:

$$\max_{\Phi} \sum_{(x,y) \in \mathcal{Z}} \sum_{t=1}^{|y|} \log(P_{\Phi}(y_t|x, y_{<t}))$$

In low rank representation, we adopt a parameter-efficient method and instead of learning a full parameter increment $\Delta\Phi$, a low dimensional task-specific parameter vector Θ is introduced as;

$$\Delta\Phi = \Delta\Phi(\Theta)$$

where $|\Theta| \ll |\Phi_0|$. The objective function then becomes:

$$\max_{\Theta} \sum_{(x,y) \in \mathcal{Z}} \sum_{t=1}^{|y|} \log(p_{\Phi_0 + \Delta\Phi(\Theta)}(y_t|x, y_{<t}))$$

3.3 Mathematical Formulation

For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, we restrict its update to a low-rank form by expressing it as

$$W_0 + \Delta W = W_0 + BA,$$

where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \leq \min(d, k)$. During training, W_0 kept fixed (i.e., it receives no gradient updates), and only A and B are trainable. Since both W_0 and $\Delta W = BA$ act on the same input, their outputs are added element-wise. Denoting $h = W_0 x$, the modified forward pass becomes

$$h = W_0 x + \Delta W x = W_0 x + BA x$$

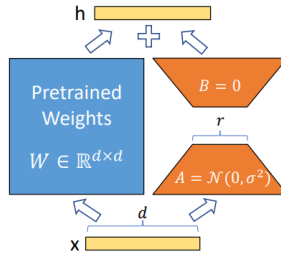


Fig 1. Reparametrization. Only A and B are trained [1]

In practice, we initialize A with random Gaussian entries and B with zeros, so that $\Delta W = BA = 0$ at the start of training. We then rescale $\Delta W x$ by a factor $\frac{\alpha}{r}$ where, α is a constant independent of r . Under Adam optimization, adjusting α is roughly equivalent to adjusting the learning rate,

provided the initialization is scaled appropriately. Consequently, we fix α to the first chosen value of r and do not tune it further.

3.4 Application of LoRA to Transformer

In principle, LoRA can be applied to any subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, the self-attention block contains four weight matrices (W_q, W_k, W_v, W_o), and the MLP block contains two additional weight matrices. For simplicity, we treat W_q (and similarly W_k, W_v) as a single matrix of size $d_{\text{model}} \times d_{\text{model}}$.

LoRA introduces additional trainable low-rank matrices that are placed in parallel with existing weight matrices. The total number of trainable parameters contributed by LoRA depends on the chosen rank r and the dimensions of the the original weight matrices, and is given by:

$|\Theta| = 2 \times \widehat{L_{\text{LoRA}}} \times d_{\text{model}} \times r$, where $\widehat{L_{\text{LoRA}}}$ denotes the number of weight matrices to which LoRA is applied. In this work, we inject LoRA weights to the attention weight matrices and the MLP modules. We further examine the impact of applying LoRA to different layers in the Transformer model like GPT-2.

4 Experimental Set-up

We evaluate the performance of LoRA using GPT-2 [4] on an SMS spam classification dataset. As a baseline, we first measure performance using full fine-tuning of all layers, where the model is initialized with pre-trained weights and biases, and every parameter is updated via gradient descent. We then consider a variant in which only a subset of layers is updated while the remaining layers are frozen. In this report, we summarize and compare the performance metrics obtained from full fine-tuning, selective layer updating, and the LoRA-based adaptation approach. Through these experiments we want to answer:

- 1) Given parameter budget constraints, which set of weight matrices could be adapted in a pre-trained Transformer for efficient and maximum downstream performance task (classification task)
- 2) Which layer in Transformer outputs the best performance metrics
- 3) What is the optimal rank r for LoRA?

GPT2 configuration

```
BASELINE_CONFIG = {
    "vocab_size": 50257,
    "context_length": 1024,
    "drop_rate": 0.0,
    "qkv_bias": True
}
```

Model	Trainable parameters (M)	Emb_dim	Number of layers	Number of heads
gpt2-small	124	768	12	12
gpt2-medium	355	1024	24	16
gpt2-large	774	1280	36	20
gpt2-xl	1558	1600	48	25

SMS dataset Label	Total instances	Balanced dataset instances
Ham (0)	4825	747
Spam(1)	747	747

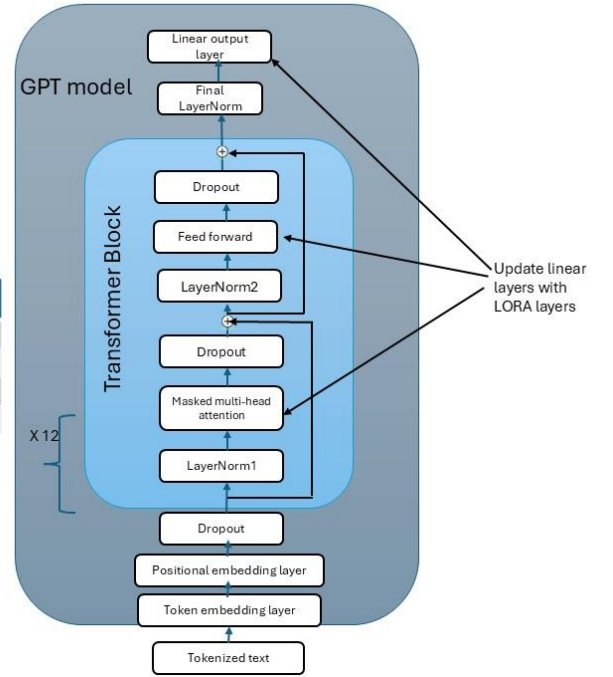


Fig 2. Various GPT-2 model configurations have been shown (Left); Application of LoRA layers to the Transformer blocks and linear layer (Right)

Trainable parameters	124M
Optimizer	Adam
Dropout Prob	0
Batch Size	8
# Epoch	20
Learning Rate	5e-05
rank	4,8,16,32
LoRA α	4,816,32

Table 1. The hyperparameters used for training GPT-2 pre-trained model with LoRA.

5 Result & Analysis

We varied rank, scale, and trainable layers and recorded the training, validation, test accuracy and losses. The results from empirical experiments have been summarized in the below table.

rank	alpha	inable_lay	lr	weight_decay	num_epochs	final_train_acc	final_val_acc	final_test_acc	train_loss_last	val_loss_last	examples_seen	minutes
4	4	lora	5e-05	0.1	20	1	0.9731543624161074	0.9533333333333334	0.001471168	0.1790742494953232	20800	3.521325492858886
4	8	lora	5e-05	0.1	20	1	0.9664429530201344	0.95	2.8982450485273147e-05	0.2113334769975201	20800	3.363683108488719
4	16	lora	5e-05	0.1	20	1	0.9731543624161074	0.9733333333333334	4.031732018461298e-06	0.1888704588136608	20800	3.370514730612437
4	32	lora	5e-05	0.1	20	1	0.9865771812080536	0.9533333333333334	4.241635356720508e-05	0.1082600627122175	20800	3.345338591
8	4	lora	5e-05	0.1	20	1	0.953020134	0.9533333333333334	2.12919196474104e-05	0.2727795317887285	20800	3.3892415245374043
8	8	lora	5e-05	0.1	20	1	0.9664429530201344	0.97	1.95685644436594e-05	0.1999975365915756	20800	3.4824103832244875
8	16	lora	5e-05	0.1	20	1	0.9865771812080536	0.96	1.1328299979140866e-05	0.1296150168874838	20800	3.484909995936932
8	32	lora	5e-05	0.1	20	0.9932692307692308	0.9731543624161074	0.95	0.028532659	0.054427129	20800	3.687904656
16	4	lora	5e-05	0.1	20	1	0.9731543624161074	0.9666666666666668	3.782378066944148e-05	0.1408333134769464	20800	3.393565634886424
16	8	lora	5e-05	0.1	20	1	0.9798657718120806	0.97	1.5348532741654708e-05	0.093082455	20800	3.4344970107078554
16	16	lora	5e-05	0.1	20	1	0.9798657718120806	0.95	3.526347640217864e-05	0.1868661438077556	20800	3.362508726119996
16	32	lora	5e-05	0.1	20	0.9980769230769232	0.9798657718120806	0.9433333333333334	0.006153102	0.1695181633603122	20800	3.454322640101115
32	4	lora	5e-05	0.1	20	1	0.9731543624161074	0.96	9.33959432586562e-06	0.2135916940479116	20800	3.368599224090576
32	8	lora	5e-05	0.1	20	1	0.9798657718120806	0.98	9.883591087600508e-06	0.1622889810252589	20800	3.426905175
32	16	lora	5e-05	0.1	20	1	0.9865771812080536	0.9733333333333334	2.0871252972938235e-05	0.1302207562253227	20800	3.493814214070638
32	32	lora	5e-05	0.1	20	0.9990384615384615	0.9664429530201343	0.9433333333333334	0.002605155	0.1492961513347262	20800	3.407451308

Table 2. Training, validation, test accuracy and loss across different configurations of rank and alpha with trainable layers as LoRA.

trainable_layers	lr	weight_decay	num_epochs	final_train_acc	final_val_acc	final_test_acc	train_loss_last	val_loss_last	examples_seen	minutes
all	5e-05	0.1	20	1	0.9798657718120806	0.9633333333333334	7.625853481272315e-06	0.2240119241366636	20800	3.489137589931488
last_block	5e-05	0.1	20	0.9971153846153846	0.9731543624161074	0.97	0.004521322	0.1006043753639364	20800	1.967237369219462
last_layer	5e-05	0.1	20	0.8682692307692308	0.8322147651006712	0.8733333333333333	0.5037569975852967	0.5440738687389776	20800	1.8457330346107483

Table 3. Training, validation, test accuracy, and loss across different trainable layers.

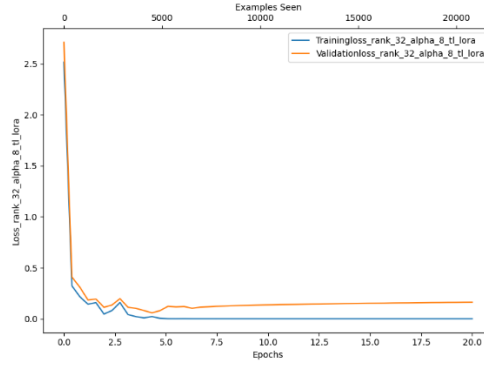


Fig 3. Loss curve for rank = 32 and alpha = 8; trainable layer is LoRA

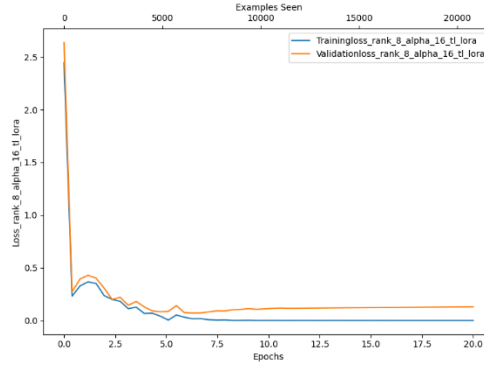


Fig 4. Loss curve for rank = 8 and alpha = 16; trainable layer is LoRA

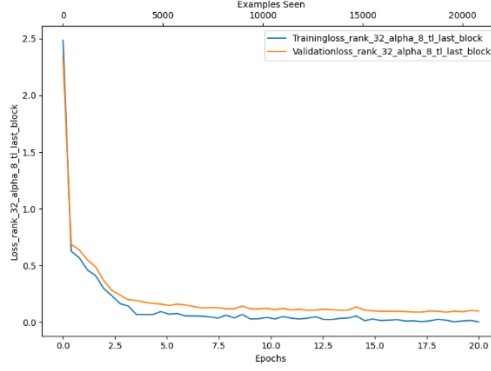


Fig 5. Loss curve when the trainable layer is last transformer block

Table 2 reports the outcomes of the LoRA hyperparameter sweep over rank $r \in \{4, 8, 16, 32\}$ and scaling factor $\alpha \in \{4, 8, 16, 32\}$. All configurations share identical optimization settings: learning rate $5e-05$, weight decay 0.1 and 20 training epochs. Across all runs, the model attains essentially perfect training accuracy, with final training accuracy very close to 1.0 and correspondingly small training losses. This indicates that, even at the lowest ranks, the LoRA adapters provide sufficient capacity to fit the training data.

Validation and test performance are consistently high. Depending on the choice of r and α , validation accuracy lies in the range of 0.95–0.99, and test accuracy ranges from approximately 0.94 to 0.98. The best configuration in terms of held-out performance is obtained for rank $r=32$ and scaling factor $\alpha=8$. This setting achieves a test accuracy of 0.98 and a validation accuracy of approximately 0.98, together with low training and validation losses, making it the strongest LoRA model in the sweep. Several other configurations are competitive; for example, $r=16$, $\alpha=8$ and $r=4$, $\alpha=16$ both yield test accuracies of at least 0.97, demonstrating that high performance can be achieved with considerably smaller adapter ranks.

Table 3 presents complementary baselines in which LoRA is not used, and different subsets of the base model parameters are made trainable: all layers, only the last Transformer block, and only the final output layer. Full fine-tuning of all layers achieves perfect training accuracy, a validation accuracy of 0.98, and a test accuracy of 0.96. Training only the last block yields a training accuracy of 0.997, validation accuracy of 0.973, and test accuracy of 0.97, while updating only the final layer results in noticeably lower performance, with validation and test accuracy around 0.83 and 0.87, respectively. Training times are in the order of 3.5 minutes for full fine-tuning and LoRA, and around 2 minutes when only the last block or last layer is updated.

Overall, the quantitative results show that LoRA with carefully chosen hyperparameters, can match or exceed the accuracy of full-model fine-tuning, while more restricted adaptation strategies such as training only the last layer fall substantially short.

6. Conclusions

The LoRA sweep reveals several consistent patterns regarding capacity, generalization, and overfitting. Because almost all configurations reach training accuracy close to 1.0 with very small training losses, differences in performance are driven primarily by generalization rather than by optimization failure. For a fixed α , increasing the rank from 4 to 16 typically improves validation and test accuracy, but the gains diminish as the rank grows further to 32. This suggests that moderate ranks already capture most of the beneficial variations introduced by LoRA, and that very high ranks offer only limited additional benefits on this dataset.

The scaling factor α emerges as a critical hyperparameter. Intermediate values ($\alpha=8$ or $\alpha=16$) tend to produce the best balance between fitting the data and avoiding overfitting. Very small α sometimes underutilizes the adapter capacity, leading to slightly lower validation and test scores, whereas very large $\alpha=32$ at higher ranks (16 and 32) yields clear signs of overfitting. In those cases, training and validation accuracies remain high, but test accuracy drops around 0.94, and the train–test gap widens. The best model, with $r=32$ and $\alpha=8$, maintains both near-perfect training accuracy and the highest test accuracy observed, with a small train–test discrepancy, indicating good generalization.

Comparison with alternative trainable-layer configurations clarifies the role of LoRA relative to conventional fine-tuning. Full fine-tuning of all parameters gives strong performance but slightly worse test accuracy than the best LoRA configuration, despite updating vastly more parameters. This suggests that low-rank adaptation can act as an implicit regularizer, constraining the effective search space and thereby improving generalization. Fine-tuning only the last Transformer block performs surprisingly well, reaching a test accuracy of 0.97 with reduced training time and parameter count; this indicates that a significant portion of task-specific adaptation can be realized in the terminal block of the network. In contrast, updating only the final classification layer leads to clear underfitting, both the training and validation losses remain relatively high, and the model fails to fully exploit the representational capacity of the frozen backbone.

Taken together, these findings indicate that LoRA provides a principled and effective mechanism for parameter-efficient adaptation. With appropriately selected rank and scaling, LoRA not only matches but can surpass the performance of full fine-tuning, while using far fewer trainable parameters. At the same time, the comparison with last-block and last-layer baselines shows that merely adapting the output head is insufficient, and that meaningful gains arise from modifying deeper transformer block either directly, as in the last-block setting, or via structured low-rank updates, as in LoRA. Further, LoRA achieves these accuracy levels with a substantially smaller number of trainable parameters, which directly reduces the memory footprint and computational cost of optimization. Consequently, training converges in less wall-clock time while remaining feasible on more modest hardware, making LoRA a scalable adaptation strategy for increasingly large base models and datasets.

References

- [1] Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685.
- [2] Aghajanyan, Armen, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics
- [3] Houlsby, Neil, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. *In Proceedings of the 36th International Conference on Machine Learning*, pages 2790–2799, Long Beach, California. PMLR.
- [4] Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI, San Francisco, California.
- [5] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *In Advances in Neural Information Processing Systems 30*, pages 5998–6008.