An Introduction to LoRA

Sanjana Surapaneni, Nicholas Croteau, Kerui Wu

November 2024

1 Introduction

1.1 What is LoRA?

LoRA, or Low-Rank Adaptation, is a technique that greatly reduces the number of trainable parameters in a model to for the purpose of lowering compute-resources. This is especially beneficial when a model needs retraining between different tasks. To introduce why LoRA is an excellent technique, it is helpful first to explore two prominent machine learning strategies: transfer learning and fine-tuning.

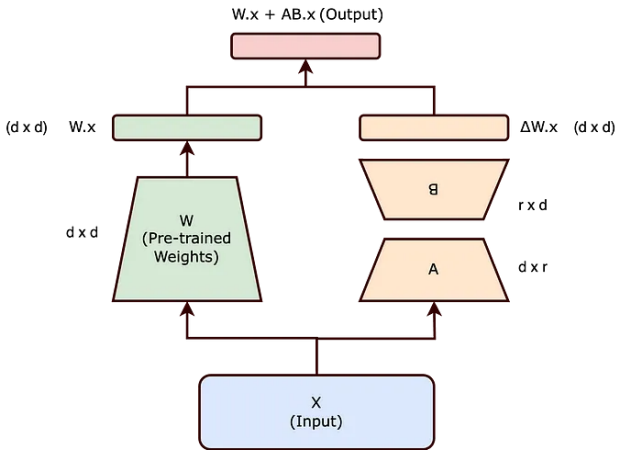

Figure 1: The LoRA process includes freezing the initial weights (green), extracting two smaller-sized matrices (orange) from the weight matrix, and combining after fine-tuning the A and B matrices to produce the output (red).

Transfer learning involves leveraging knowledge from a pre-trained model and applying it to a related but distinct problem, accelerating training and improving performance with limited data. On the other hand, fine-tuning refines a

pre-trained model by retraining specific layers or parameters to optimize performance on a new task. As previously mentioned, LoRA reduces computational resource demands by minimizing storage requirements during each retraining step for every layer in the network, making it an effective complement to these methods that may be computationally expensive.

1.2 A review of rank

If you remember from Linear Algebra, the rank of a matrix refers to the maximum number of linearly independent rows or columns it possesses. Another way to envision rank is that linearly independent vectors within the matrix cannot be expressed as a linear combination of other vectors in the set.

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 6 & 9 \\ 2 & 4 & 6 \end{bmatrix}$$

Figure 2: Matrix with rank 1, as the second and third rows are scalar multiples of the first row (multiplied by 3 and 2, respectively).

This is useful because the matrices can now be represented by a small number of linearly independent columns. Essentially, we are narrowing down the important information by removing the redundancy of linear combinations from the other columns. A high-dimensional matrix can have a small rank or be represented by a small number of columns.

2 Alternative Methods

Before looking into LoRa, there are multiple other techniques used to achieve similar results, which we will detail below. This is only a surface level detailing of each technique, but it is worth to consider to realize the need and power of LoRa in contrast to its contemporaries. The majority of the following techniques are covered in the original paper [6] as a reference to the SOTA at the time that LoRa is introduced.

2.1 Fine Tuning

Fine tuning is by far the most widely used method of those discussed, but also the most compute- and memory-intensive. Fine tuning is the continued training of parameters for a specific new task. This in effect creates a new model suited to only this new task. Therefore, if you had multiple tasks, you would have to train a given model for each given task.

Fine tuning comes with its own downsides other than just computational; a fine tuned model is prone to over fitting and forgetting general knowledge, a downside of adjusting all weights in a model.

2.2 Prefix-Tuning, Prompt Tuning

This group of techniques focuses on adding trainable tokens or learnable prefixes to a models input to steer output as desired. This can be manually playing with prompts or a system that appends certain prompts and inputs to specific tasks. E.g, we are generating colored pencil art, append 'traditional media' to the input. However, performance non-monotonically changes, that is to say, it is nebulous and unreliable to an extent, as tweaking prompts is more of an art than a science. There is also the issue of scalability, as a model at large cannot be changed just by tweaking its inputs for one task or output.

2.3 Adaptor Layers

This technique has the idea to add extra NN layers into a given architecture that are only trained on the new task desired. Expectantly, it has a low number of parameters compared to fine tuning. The problem with adaptor layers is that the added layers introduce latency. New layers similarly will increase computation and memory intensity, as well as the overall parameter count of using this method.

2.4 Quantization

Quantization reduces the precision of parameters in a model. E.g, 32-bit floats to 8 bit or lower. This expectantly reduces the memory footprint and speeds up computation depending on the degree of quantization. Conversely, with each degree of precision lost, you are potentially losing accuracy. This technique also doesn't actually redeploy a given architecture for a new task, it is only used to help speed up and compress the size of models. An interesting factoid is that it's highly likely that 4-bit quantization is universally optimal level of precision for most tasks. [2].

We will see now how LoRa addresses the above shortcomings and why it was such a breakthrough in the field of re-purposing models.

3 LoRA Operation

The following is derived from the original LoRa paper [6].

3.1 Standard Linear Transformation

Starting with a typical transformation, consider a weight matrix $W \in \mathbb{R}^{d \times k}$ that is part of a linear transformation in a layer of a neural network, where d

is the input dimension and k is the output dimension. Given an input vector $x \in \mathbb{R}^d$, the output of the linear transformation is:

$$y = Wx$$

3.2 Adding a Low-Rank Adaptation

To adapt the model to a new task without modifying W directly, we introduce a low-rank matrix ΔW as a trainable update to W. This gives us the modified weight matrix:

$$W' = W + \Delta W$$

where W remains fixed, and only ΔW is updated during the tuning.

3.3 Low-Rank Decomposition of ΔW

The matrix $\Delta W \in \mathbb{R}^{d \times k}$ can be decomposed as a product of two smaller matrices:

$$\Delta W = BA$$

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

Thus, the adapted weight matrix becomes:

$$W' = W + BA$$

This decomposition reduces the number of trainable parameters from $d \times k$ (in the full-rank case) to $r(d + k)$ in the low-rank case, which is much smaller when r is small.

3.4 Forward Pass with the Adapted Weight Matrix

In the forward pass, we use the modified weight matrix W' instead of W:

$$y = W'x = (W + BA)x$$

Expanding this, we get:

$$y = Wx + B(Ax)$$

Here, Wx represents the original, pretrained transformation, $B(Ax)$ is the low-rank adaptation specific to the new task.

3.5 Training Only the Low-Rank Adaptation

During training on the new task, we freeze W and update only B and A. This low-rank adaptation captures the task-specific information while leaving the original model parameters unchanged, leading to a parameter-efficient and memory-efficient fine-tuning process.

This approach enables effective task-specific adaptation with a small number of trainable parameters $r(d + k)$, thus making it an efficient alternative to fine-tuning and the other methods discussed.

4 Potential Applications

LoRA's strong adaptability and efficiency have significantly improved various applications, targeting transfer learning and improving model performance, including accuracy, scalability, and robustness. In this section, we will briefly introduce some of its applications in Natural Language Processing(NLP) and Computer Vision(CV) fields based on the survey [10].

4.1 Natural Language Processing

The rapid advancements in pre-trained language models, especially large language models (LLMs), are transforming approaches to language tasks due to their impressive performance. Despite being trained on vast amounts of general data, these models still need further fine-tuning on task-specific data to perform well in downstream tasks. Consequently, using LoRA for fine-tuning these pre-trained models is a logical choice, as it helps lower the computational resources needed.

LoRA is widely adopted in LLaMA for diverse traditional NLP tasks, such as emotion recognition [14], text classification [9], and role recognition [1]. [15] proposed a customized LoRA structure to improve the pre-trained language model's performance for Human-Centered Text Understanding(HCTU), a type of domain adaptation to transfer the pre-trained model to match the user's preference.

In addition to traditional NLP tasks, several methods have been proposed to apply LoRA to improve model performance in various code-related tasks. [12] uses LoRA to fine-tune Llama for automated program repair(APR); [11] fine-tuned with LoRA for Text-to-SQL task.

4.2 Computer Vision

In vision tasks, LoRA is primarily applied to image generation and image segmentation. In dealing with image generation tasks, LoRA is widely used in diffusion models to address various image generation tasks while reducing computational resources. [8, 4] use LoRA to fine-tune diffusion models for image style transfer, while [7] apply LoRA to text-to-image generation. Similar to

[15], which personalizes LoRA to improve the model's performance, [5] proposes Smooth Diffusion, which uses LoRA to achieve smoothness in the latent space, leading to better performance in various image generation tasks.

The image segmentation task aims to divide an image into multiple meaningful regions or objects. SAM has been proposed as a foundational model for image segmentation and demonstrated superior generalization ability. [3] propose SamLP, which utilizes LoRA to adapt SAM for efficient segmentation of license plates. [13] fine-tunes SAM's encoder using LoRA for instance segmentation task, which can in turn be used in structural damage detection.

5 Empirical Results

5.1 Image Generation

Here we will display side by side comparisons of computational expenses of Image Generation with and without the addition of LoRA.

5.2 Generation Environment

The following section covers the specifications for which the empirical testing on LoRa was performed on, and the results therein. A word of warning, image-generation set-ups are much like a musician and their hurdy-gurdy; that is to say that each one has been tinkered with and has matured as a tool alongside the artist, and that replicating the exact conditions of the environment can prove difficult to impossible. Regardless the conditions of the environment will be listed below, but many minor settings will be omitted for brevity as it is not the focus of the test.

As stable-diffusion utilizes all available system V-RAM, testing the hardware usage is not applicable. The number of parameters altered can vary between LoRa to LoRa, and are usually reflected in the size (in mBs) of a LoRa file. Therefore the only suitable way to measure LoRa impact is the time it takes to generate, with and without a LoRa attached. As stated previously, the specific time is highly variable on the machine and the generation environment, but we should be able to see a clear difference or lack thereof in time-spent per generation relative to LoRa and non-LoRa generation.

Parameter	**Value**
Stable Diffusion Version	1.10.1
Python Version	3.10.6
Torch Version	2.01+cu118
Xformers Version	0.0.20
Gradio Version	3.41.2
Checkpoint Hash	821aa5537f (SDXL Pony Diffusion)
Clip Skip	2

SD VAE	sdxl_vae.safetensors
Sampling Method	Euler A
Schedule Type	Uniform
Sampling Steps	20
Image Width	1024 pixels
Image Height	1500 pixels
CFG Scale	7
Seed	536003248
Negative Prompts	None
Positive Prompts	Variable (discussed below)
Eta	1
Sigma Noise	1

Note that this is a typical environment that end-users will use and it is fairly SOTA. Most users will be applying some form of cross attention optimization, in our case, xformers; this will speedup our times at the cost of quality. The specifics of all the effects of the above on the generation of images is beyond the scope of this text. There is also a method that stable diffusion employs to batch generate images to reduce computation time, but for simplicity we will be generating images back to back as singletons.

5.3 Description of Tests

As described earlier, the primary factor under consideration is time. For each test, we generated 12 images of a given prompt, with and without the application of a LoRa, we repeated this 10 times for prompt 1. All other settings and parameters were held constant, including the random seed. This ensures reproducibility—if the same test is run multiple times, the generated outputs will remain identical.

5.4 Result of Tests

- **Prompt 1:** A well-known character, Son Goku from the Dragon Ball series. This character is expected to be well-represented in most models due to the abundance of existing media featuring Goku.
- **Prompt 2:** Henrietta, a lesser-known character from the anime Gunslinger Girl. This prompt evaluates the model's performance on a character with comparatively limited media representation.

The exact prompts used for the tests are as follows:

- **Prompt 1 (Goku):** `score_8_up, score_7_up, ((Solo:1.0)), masterpiece, best quality, highres, simple background, sfw, muscles, 1boy, face focus, son goku, super saiyan, <lora:Goku_Pony_XL:1>`

- **Prompt 2 (Henrietta):** `score_8_up, score_7_up, solo, masterpiece, best quality, highres, simple background, sfw, 1girl, blazer, brown eyes, brown hair, collared shirt, cowboy shot, neck ribbon, henrietta (gunslinger_girl), <lora:Henrietta:1>`

The only difference between the prompts with and without LoRa is the addition of the last token, signaling to the model to mount the LoRa.
From a purely subjective standpoint, we can see that the addition of the LoRa more accurately reflects the character we want to depict, i.e the quality of the output is higher, consistently so. The time spent generating however was not wildly different, as detailed below.

Generation Times with and without LoRAs

Prompt	Without LoRAs (Time)	With LoRAs (Time)
Goku	4:44.2	4:44.1
Goku	4:43.9	4:52.9
Goku	4:47.5	4:45.2
Goku	4:46.3	4:48.6
Goku	4:45.8	4:50.3
Goku	4:44.7	4:46.9
Goku	4:46.1	4:47.4
Goku	4:45.3	4:49.1
Goku	4:46.8	4:48.2
Goku	4:45.0	4:47.6
Henrietta	0:25.7	0:26.1

Table 2: Measured generation times for images with and without LoRAs across two prompts.

The times were not different enough to conclude that LoRa adds any overhead on the scale of a batch of 12 images. This is to say, that the difference in computation time can be attributed to the latent noise of the hardware's availability; In a typical user's machine, several other processes will start and stop, impacting performance. In a professional environment, we may be able to see clearly that LoRa has little to no effect on computation time because of a more stable availability of GPU resources.

(a) Without LoRAs - Prompt 1

(b) With LoRAs - Prompt 1

(a) Without LoRAs - Prompt 2

(b) With LoRA - Prompt 2

Figure 4: Comparison of generated art with and without LoRAs for Prompt 2.

We conclude our empirical testing in this section by stating that LoRa has little to no impact on computational time, while offering significantly higher quality.

6 Text Generation

References

[1] Tobias Bornheim, Niklas Grieger, Patrick Gustav Blaneck, and Stephan Bialonski. Speaker attribution in german parliamentary debates with qlora-adapted large language models. *arXiv preprint arXiv:2309.09902*, 2023.

[2] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws, 2023.

[3] Haoxuan Ding, Junyu Gao, Yuan Yuan, and Qi Wang. Samlp: A customized segment anything model for license plate detection. *arXiv preprint arXiv:2401.06374*, 2024.

[4] Yarden Frenkel, Yael Vinker, Ariel Shamir, and Daniel Cohen-Or. Implicit style-content separation using b-lora. *arXiv preprint arXiv:2403.14572*, 2024.

[5] Jiayi Guo, Xingqian Xu, Yifan Pu, Zanlin Ni, Chaofei Wang, Manushree Vasu, Shiji Song, Gao Huang, and Humphrey Shi. Smooth diffusion: Crafting smooth latent spaces in diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7548–7558, 2024.

[6] 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 preprint arXiv:2106.09685*, 2021.

[7] Likun Li, Haoqi Zeng, Changpeng Yang, Haozhe Jia, and Di Xu. Block-wise lora: Revisiting fine-grained lora for effective personalization and stylization in text-to-image generation. *arXiv preprint arXiv:2403.07500*, 2024.

[8] Shaoxu Li. Diffstyler: Diffusion-based localized image style transfer. *arXiv preprint arXiv:2403.18461*, 2024.

[9] Zongxi Li, Xianming Li, Yuzhang Liu, Haoran Xie, Jing Li, Fu-lee Wang, Qing Li, and Xiaoqin Zhong. Label supervised llama finetuning. *arXiv preprint arXiv:2310.01208*, 2023.

[10] Yuren Mao, Yuhang Ge, Yijiang Fan, Wenyi Xu, Yu Mi, Zhonghao Hu, and Yunjun Gao. A survey on lora of large language models. *arXiv preprint arXiv:2407.11046*, 2024.

[11] Richard Roberson, Gowtham Kaki, and Ashutosh Trivedi. Analyzing the effectiveness of large language models on text-to-sql synthesis. *arXiv preprint arXiv:2401.12379*, 2024.

[12] André Silva, Sen Fang, and Martin Monperrus. Repairllama: Efficient representations and fine-tuned adapters for program repair. *arXiv preprint arXiv:2312.15698*, 2023.

[13] Zehao Ye, Lucy Lovell, Asaad Faramarzi, and Jelena Ninic. Sam-based instance segmentation models for the automation of masonry crack detection. *arXiv preprint arXiv:2401.15266*, 2024.

[14] Yazhou Zhang, Mengyao Wang, Prayag Tiwari, Qiuchi Li, Benyou Wang, and Jing Qin. Dialoguellm: Context and emotion knowledge-tuned llama models for emotion recognition in conversations. *arXiv preprint arXiv:2310.11374*, 2023.

[15] You Zhang, Jin Wang, Liang-Chih Yu, Dan Xu, and Xuejie Zhang. Personalized lora for human-centered text understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19588–19596, 2024.