Task 1.2.2 - Cellula Technologies NLP Internship
Efficient Fine-Tuning, LoRA and QLoRA
Short Research Article

Abdelrahman Elaraby

September 2025

Contents

1 Introduction

Deep Neural Network Models have become the foundation of many Artificial Intelligence tasks, whether in Computer Vision or Natural Language Processing (NLP). For example, Large Language Models (LLMs) enabled many breakthroughs and proved to be effective across a wide range of tasks. However, these large models come with a great cost, which is that they are extremely resource-intensive.
Fine-Tuning is the process of taking a Pre-trained Model on a very large diverse general-purpose dataset and tweaking (training) its parameters to do well on a specific tasks with small datasets. But doing Full Fine-Tuning on such huge models requires updating all parameters and often consumes hundreds of gigabytes of GPU memory and weeks of training time on specialized hardware. That problem derived the research community to respond with a variety of parameter-efficient fine-tuning (PEFT) strategies that allow models to be adapted to new tasks while only modifying a small fraction of their parameters or adding few extra parameters to be modified, which are called Adapters. Among those influential PEFT methods are LoRA (Low-Rank Adaptation) and its extension QLoRA (Quantized LoRA) which will be the topic of this short research article.

2 Background

2.1 Parameter-Efficient Fine-Tuning (PEFT)

Parameter-efficient fine-tuning refers to the methods that tries to fine-tune/modify only a small number of parameters compared to the size of the whole model. And to elaborate more, it is the training process that introduces huge number of parameters in addition to the model parameters themselves, those are the gradient and the optimizer states (like momentum and variance in Adam). So, although the number of parameters of the original model isn't reduced by those PEFT methods, the training becomes more efficient when only a small percentage of the parameters are trained.
In order to illustrate the enormous requirements of full fine-tuning, consider a 7B parameter model. Using standard full fine-tuning in half precision (fp16), the parameters themselves require approximately 14 GB of GPU memory, while gradients and Adam optimizer states add an additional 14 GB and 28 GB respectively, for a total of about 56 GB. This makes full fine-tuning impractical on a single GPU.

2.2 Adapters

Adapters are small neural modules inserted into a pre-trained model between its layers, and those inserted modules are the only modules to be fine-tuned in the model. There are multiple variations of adapter-based methods each with its own trade-offs; some of them are:

- **Standard Adapters:** Introduced by Houlsby et al. (2019), these consist of a down-projection, nonlinearity, and up-projection. Only the adapter parameters are trained while the backbone model remains frozen.
- **Prefix-Tuning:** Prepends learnable vectors (prefixes) to each transformer block, steering the model behavior without modifying its main parameters.
- **Prompt-Tuning and P-Tuning:** Learns soft prompt embeddings at the input layer (or across multiple layers in P-Tuning v2), which guide the model similarly to textual prompts.

Although these methods significantly reduce training costs, most of them still involve introducing additional modules into the model's forward pass. This can add inference overhead and architectural complexity. To address these limitations, **LoRA (Low-Rank Adaptation)** was introduced.

3 LoRA (Low-Rank Adaptation)

3.1 Concept and Methodology

LoRA (Low-Rank Adaptation) is a parameter-efficient fine-tuning technique that modifies how we adapt/fine-tune Deep Neural Networks or Large Language Models (LLMs) to new specific tasks. Instead of inserting entire new modules as in traditional adapters, or instead of training all model parameters as in full fine-tuning, LoRA introduces a lightweight mechanism to update only a small subset of weights. The key idea is that the weight changes or updates in neural networks often happen in a low-dimensional subspace. In other words, although the eight matrix can be

very huge, the effective changes required for fine-tuning to a task can be represented and approximated with much smaller low-rank structure.

3.2 Formulation and Details

Let $W_0 \in \mathbb{R}^{d \times k}$ be a pre-trained weight matrix in a transformer model for example. Instead of updating all entries of W_0 during fine-tuning, LoRA re-parameterizes the weight update ΔW as a product of two low-rank matrices:

$$W = W_0 + \Delta W = W_0 + AB \tag{1}$$

where:

- $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$,
- $r \ll \min(d, k)$ is the chosen rank, typically a small value such as 4, 8, or 16.

This decomposition reduces the number of trainable parameters from $d \times k$ to $r(d + k)$. For large matrices, this represents a significant efficiency gain. In practice, LoRA method is commonly applied to the query and value projection matrices within the attention mechanism of the transformers, since these components play a central role in shaping the model's contextual understanding. A visual illustration is shown in Figure 1

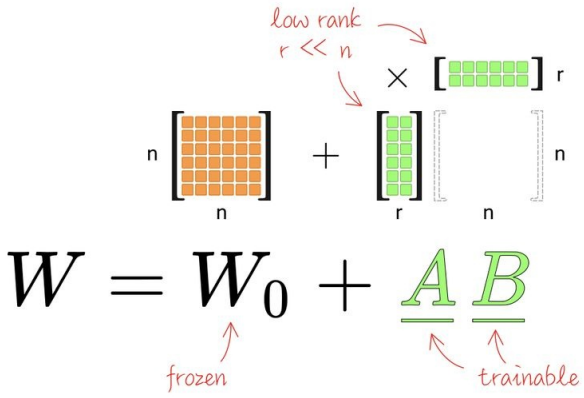

Figure 1: Illustration of LoRA: instead of updating the full weight matrix, low-rank matrices A and B capture task-specific adaptations. Image source: Towards Data Science.

Then after those small matrices are fine-tuned, they can be actually added and merged to the model weights to end up with a final fine-tuned model. The rule of thumb for choosing the rank "r" is that if the new task we need to fine-tune on it is a bit far from the learned original task the model was pre-trained on then make "r" bigger. a bigger "r" means more capacity to adjust the frozen model.

3.3 Numerical example

In contrast to the numerical example mentioned in the Background section about the full fine-tuning of the 7B parameter model, LoRA freezes the base model and introduces only $\sim$ 60M trainable parameters (for rank $r = 8$), requiring about 120 MB for parameters, 120 MB for gradients, and 240 MB for optimizer states. Together with the frozen 14 GB fp16 base model, the total training memory is reduced to roughly 14.5 GB.

4 QLoRA (Quantized Low-Rank Adaptation)

Although LoRA significantly reduces the number of trainable parameters, the base model still consumes massive GPU memory because it is stored in full precision (e.g., FP16 or BF16). So, huge LLMs may still be resource-intensive to fine-tune. QLoRA (Quantized LoRA) was introduced to solve this problem by combining LoRA with the quantization techniques. So, the idea is to load the frozen model in a very compressed format like 4-bit quantization, while still training LoRA adapters with their high precision. This drastically lowers the needed memory while training. It was worth mentioning that there is a de-quantization step done "on the fly" to these quantized frozen model weight matrices one-by-one during the forward pass (to add them to the full-precision LoRA adapters), and then they are discarded. Although this may add time overhead, it is an accepted tradeoff because the main bottleneck is GPU memory. After fine-tuning is done, you can keep the model quantized or de-quantize it to its original size and add the adapters to it.
And continuing on the numerical example of training 7B parameter model illustrated earlier; QLoRA pushes this further by storing the frozen base model in 4-bit quantized form, reducing its footprint to about 3.5 GB while keeping the LoRA adapters in fp16. The resulting total training memory is only ~ 4 GB, enabling fine-tuning of very large models on modest hardware.

5 Advantages and Limitations

LoRA proposed a major leap forward in parameter-efficient fine-tuning (PEFT) by allowing low-rank matrices to be trained while keeping the original model weights frozen. This drastically reduces the number of trainable parameters, makes training faster, and enables modularity, since different LoRA adapters can be fine-tuned for various tasks and swapped in or merged with the base model whenever needed. However, LoRA still requires the base model to be stored in full precision which of course poses a challenge for very huge models. In addition, its effectiveness depends heavily on the chosen rank, where too low may lead to underfitting, while too high reduces the efficiency gains.
QLoRA builds upon these strengths while directly addressing LoRA's main weakness which is the memory footprint of the base model. QLoRA quantizes the backbone model into 4-bit precision using techniques such as NF4 quantization and double quantization, QLoRA makes it possible to fine-tune models with tens of billions of parameters on a single GPU. Nevertheless, QLoRA introduces its own trade-offs. Quantization adds some computational overhead during training and inference, and although NF4 preserves accuracy well, certain sensitive tasks may still exhibit minor performance degradation compared to higher-precision methods. Additionally, QLoRA is more complex to implement, requiring specialized libraries such as bitsandbytes and Hugging Face's PEFT to function effectively.

6 Conclusion

So to sum things up, LoRA and QLoRA represent two of the most impactful developments in parameter-efficient fine-tuning (PEFT) of deep learning models. LoRA proved that task-specific adaptations could be done with only a fraction of the trainable parameters by introducing low-rank updates compared to the full resource-intensive fine-tuning. QLoRA took this one step further by applying quantization to the frozen backbone model allowing even the largest models to be fine-tuned on modest hardware without significant accuracy loss. Together, those approaches lower the computational barrier to working with advanced Deep Learning Models such as LLMs.

References

[1] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, B. Li, L. Wang, and W. Chen, *LoRA: Low-Rank Adaptation of Large Language Models*, arXiv:2106.09685, 2021.

[2] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Lewis, *QLoRA: Efficient Finetuning of Quantized LLMs*, arXiv:2305.14314, 2023.

[3] N. Houlsby, A. Giurgiu, S. Jastrzebski, et al., *Parameter-Efficient Transfer Learning for NLP*, Proceedings of ICML, 2019.

[4] H. Zhang, *Parameter-Efficient Fine-Tuning for NLP Models*, Medium (Hugging Face), 2023. https://medium.com/huggingface/distilbert-8cf3380435b5

[5] Hugging Face, *PEFT: Parameter-Efficient Fine-Tuning Library*, Documentation, 2023. https://huggingface.co/docs/peft

[6] T. Mikolov, A. Joulin, and E. Grave, *Efficient Training of Word Representations in Vector Space*, arXiv:1301.3781, 2013.