A Research on LoRA and QLoRA

Raghad Mohamed

September, 2025

Introduction

Large Language Models (LLMs) are called “large” for two reasons, they have an enormous number of parameters and because they are trained on massive datasets.

Fine-tuning, which is the process of adapting these pre-trained models to new domain-specific data, by adjusting these enormous numbers of parameters to better reflect the new dataset. Much like how our brains adapt when learning new ideas, fine-tuning enables models to expand their knowledge beyond their pre-training.

The challenge is that modern LLMs can contain millions or even billions of parameters. Fine-tuning all of them is prohibitively expensive in terms of GPU resources, time, and memory. This motivates new approaches for efficient fine-tuning.

Two of the most widely adopted methods today are **LoRA (Low-Rank Adaptation)** and **QLoRA (Quantized Low-Rank Adaptation)**. Both allow practitioners to fine-tune massive models at a fraction of the cost, while retaining strong performance.

LoRA

Instead of fine-tuning all parameters in a large model, LoRA freezes the majority of them. Freezing means these parameters remain untouched, exactly as they were in the original pre-trained model. Only a small subset of parameters is updated.

To integrate the updated parameters back, LoRA uses a clever trick from linear algebra: **matrix decomposition**.

When you hear about a “3 billion parameter model,” that means the model has a parameter matrix with 3 billion weights. LoRA begins by creating a new matrix that matches the size of this parameter matrix. At first glance, this seems like wasted memory, but LoRA avoids that by decomposing the large matrix into the product of two smaller ones: a column vector (single column) and a row vector (single row, which is just the transpose of the column).When multiplied, these two smaller matrices reconstruct a full matrix of the same size as the original parameter matrix. constructed The one column has a **rank of 1**, hence the term *low-rank*.

During fine-tuning, LoRA only updates these small low-rank matrices. Afterward, the low-rank updates are added back to the frozen original parameters through simple **matrix addition**.

In practice, this can mean training as little as **0.08% of the total parameters**, less than 1% compared to full fine-tuning. This efficiency is the reason LoRA has become so widely adopted.

Hyperparameters in LoRA

LoRA’s behavior is controlled by several hyperparameters:

- **Rank (r):** Determines the number of columns/rows in the low-rank matrices. Higher ranks capture more information but increase memory usage.
- **Alpha (scaling factor):** Decides how much weight the fine-tuned updates contribute when added back. Smaller values reduce overfitting by limiting reliance on fine-tuned parameters. Larger values amplify their influence.
- **Dropout:** A regularization technique that randomly deactivates parameters during training. It prevents overfitting, which is especially important since LoRA originally struggled with it.

QLoRA

It builds directly on LoRA’s mechanism but adds a critical improvement: **quantization**, which drastically reduces memory usage.

How Quantization Works

Model parameters are stored as numbers, and computers represent numbers using data types of various sizes:

A **32-bit** floating-point number requires four bytes of memory.An **8-bit** number requires just one byte. QLoRA reduces memory by storing frozen parameters in lower precision (e.g., 16-bit or 8-bit) instead of the usual 32-bit format. This compression dramatically decreases the model's memory footprint without severely impacting accuracy.

Another way to think of quantization is as "summarizing" groups of parameters. If many parameters are similar, quantization represents them with fewer representative values, effectively compressing the parameter space while keeping essential information.

Why QLoRA Matters

With quantization, QLoRA enables fine-tuning of extremely large models using a single GPU, a task previously requiring clusters of GPUs. This breakthrough makes LLM fine-tuning accessible to researchers and developers without supercomputer-scale resources.

Conclusion

LoRA and QLoRA both address the high costs of fine-tuning large language models. **LoRA** optimizes by freezing most parameters and fine-tuning only a small low-rank subset. **QLoRA** extends LoRA by quantizing frozen parameters into lower precision, further reducing memory use and making fine-tuning feasible on modest hardware.Together, these methods embody the broader trend in AI research: finding ways to make models smaller, faster, and more efficient, while still preserving their accuracy and capabilities.

References

- Red Hat. *LoRA vs QLoRA: What's the Difference?* https://www.redhat.com/en/topics/ai/lora-vs-qlora
- Aran Komatsuzaki. *LoRA and QLoRA Explained (YouTube Video).* https://youtu.be/t1caDsMzWBk?si=Nvs3t5tPBJoWP2Te