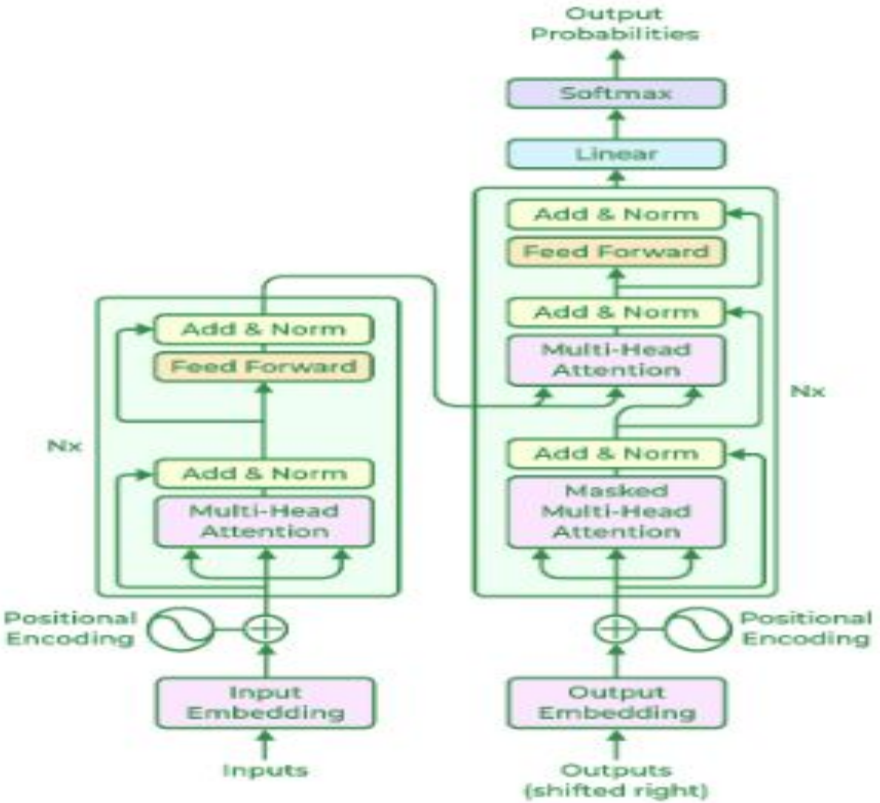
LLMs

Think of an LLM like a **very advanced, compressed archive** of the internet's text (books, websites, code, etc.).

LLM Concept	Zip File Analogy
Training data (massive texts)	The original, uncompressed data
Model training	The process of zipping (compressing) the data
Model weights (parameters)	The zipped file — compressed representation
Inference (using the model)	Unzipping just the parts you need

https://arxiv.org/pdf/1706.03762



Fine Tuning

Think of training a model like learning a new skill. You start with basic knowledge (initializing parameters), practice (forward pass), and compare your results to the target (desired output). Then, you refine your technique (backward pass) and repeat. Eventually, you become proficient (trained model).

Layer-wise Fine Tuning: This is the most commonly used method where either the early layers of the pre-trained model (closest to input) are frozen, and only the later layers are fine-tuned or the entire model is fine tuned.

Parameter Selective Fine-tuning: This method identifies and updates only a subset of parameters deemed relevant for the task.(If a transformer model has 100 million parameters, but you identify that only 10 million are highly relevant for your task , then we will finetune the latter part)

Adapter-based Fine-tuning: This technique introduces lightweight “adapter” modules alongside the

LORA(low rank Adaption)

Core Idea of LoRA

Instead of updating the original **weight matrices** (which are large), LoRA:

1. **Freezes the original model weights**.
2. **Adds small trainable low-rank matrices** to certain weight layers (usually linear layers).
3. **During training, only these low-rank matrices are updated**.

What do you mean by low rank ?

In large models like GPT or BERT, weight matrices in attention or feedforward layers are huge. For example:

- A self-attention weight matrix might be $W \in R^{4096 \times 4096}$
- Updating this directly during fine-tuning would require learning ~17 million parameters just for that layer

Empirical observations show that:

- When you fine-tune large models for specific tasks, **the actual change in the weights (ΔW)** tends to have **low intrinsic rank**.
- This means that **most of the useful adaptation** can be captured by a **low-rank matrix**, rather than a full-rank one.

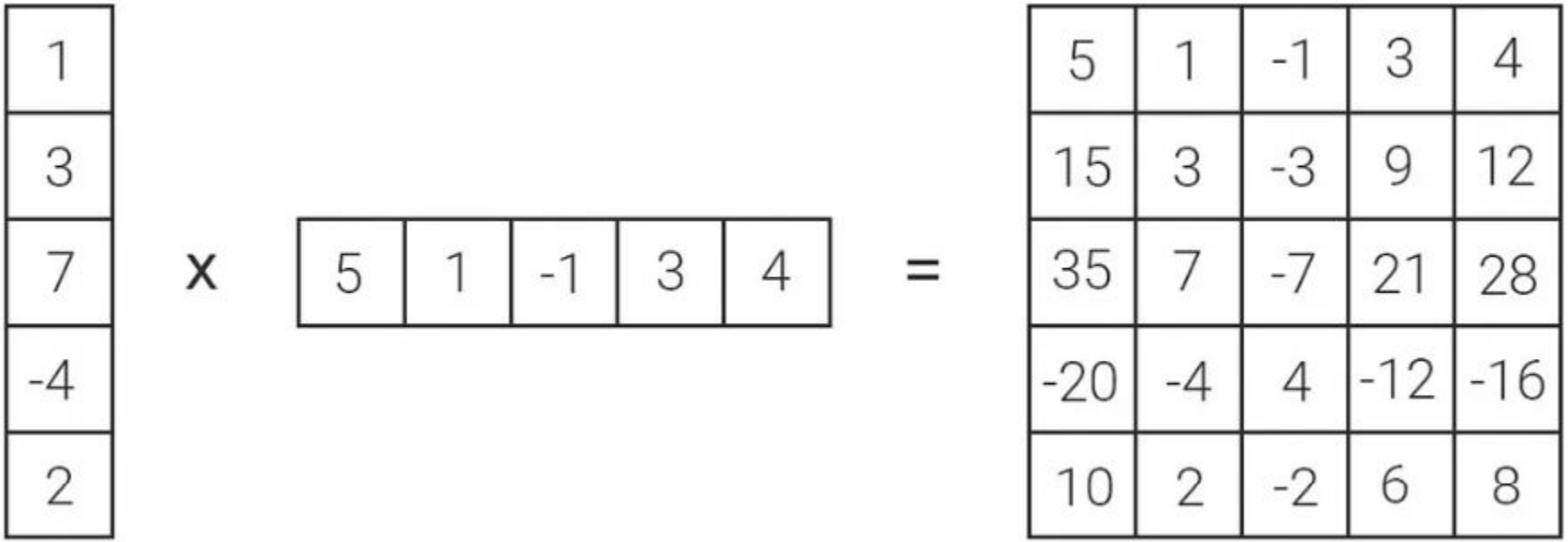
So, instead of learning a full-rank update $\Delta W \in \mathbb{R}^{d \times k}$, LoRA approximates it as:

$$\Delta W = AB, \quad A \in \mathbb{R}^{d \times r}, \quad B \in \mathbb{R}^{r \times k}, \quad \text{with } r \ll d, k$$

Why LORA ?

LoRA Shrinks and Speeds Up Fine-Tuning Through Matrix Decomposition

- This is more for conceptual understanding.Imagine a 5x5 matrix as a storage unit with 25 spaces. LORA breaks it down into two smaller matrices through matrix decomposition with “r” as rank(the dimension): a 5x1 matrix (5 spaces) and a 1x5 matrix (5 spaces). This reduces the total storage requirement from 25 to just 10, making the model more compact.
- Not only does this save space, but it also accelerates computations. Working with smaller matrices involves fewer calculations, leading to faster fine-tuning.


1
3
7
-4
2
x
5 1 -1 3 4
=
5 1 -1 3 4
15 3 -3 9 12
35 7 -7 21 28
-20 -4 4 -12 -16
10 2 -2 6 8

Strategic Focus on Attention Blocks for Maximum Efficiency

- While LORA can potentially be applied to different parts of a neural network, it's often strategically used on attention blocks within Transformer models.
- Attention blocks play a key role in LLMs, focusing on the most relevant information during language processing. By selectively adapting these blocks, LORA achieves significant efficiency gains without compromising overall performance.

How does it work ?

You add a **trainable low-rank adapter** to each selected weight matrix. Instead of updating W, you modify the forward pass like this:

$$Wx \quad \rightarrow \quad Wx + \Delta W x = Wx + (AB)x$$

Where:

- W is **frozen**
- $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$ are **trainable**
- $r \ll d, k$ → usually $r = 4, 8, 16$, depending on how efficient you want to be

Often a scaling factor α is applied:

$$\Delta W = \alpha \cdot AB$$

During fine-tuning:

- Only the **parameters of A and B** are updated.
- The pretrained weights (e.g., W) are untouched.

This drastically reduces the number of trainable parameters. For example, if $W \in \mathbb{R}^{4096 \times 4096}$ and $r = 8$, you're only training:

$$4096 \times 8 + 8 \times 4096 = 65,536 \text{ parameters}$$

Instead of ~17 million.

What is QLoRA now ??

QLoRa takes LORA a step further by quantizing the trainable parameters, representing them with fewer bits. This reduces model size even more, potentially enabling deployment on devices with limited memory and computational resources.

How does the quantization part works??

Precision:

- It refers to how accurately numbers are represented in a computer.
- Neural networks typically use 32-bit floating-point numbers, which offer high precision but can be memory-intensive.
- Example: Imagine storing the number 3.14159 as a 32-bit float.

Quantization:

- It’s a technique to reduce the size of a neural network by representing numbers with fewer bits.
- Common quantization methods include:
- Half precision (16 bits)
- Integer representation (e.g., 8 bits)
- Example: Using 16-bit half-precision for 3.14159 might store it as 3.14.

Benefits of quantization:

- Smaller models: Reduced memory usage and faster inference.
- Efficient deployment: Can run on less powerful devices like smartphones.

Trade-offs:

- Accuracy loss: Quantization can introduce small errors, potentially affecting model performance.

Example:

- Model Size: A model with 1 million weights in 32-bit floats would take 4 MB of memory.
- Using 16-bit half-precision would reduce it to 2 MB.
- Further quantizing to 8-bit integers could reduce it to 1 MB.
- Accuracy: The accuracy impact of quantization varies, but it's often minimal for many tasks.

Key Point:

- Precision and quantization are essential techniques for optimizing model size and speed while balancing potential accuracy trade-offs