LoRA + QLoRA

Low-Rank Adaptation (LoRA):

- **Parameter-efficient fine-tuning (PeFT)**: LoRA is designed to reduce the time and GPU memory required for fine-tuning large models.
- **Intrinsic dimension**: Large models contain redundant parameters, and their behavior can often be represented in a lower-dimensional space without significant loss of performance.
- **Weight update method**: During fine-tuning, the pretrained weights are frozen, and only the weight update (ΔW) is modeled as a low-rank decomposition, represented as ΔW=BA.

  ΔW\Delta W

  ΔW=BA\Delta W = B
- **Application in transformers**: LoRA modifies the attention weights by decomposing them into smaller rank matrices, significantly reducing the number of trainable parameters
- **Rank r**: The rank (dimensionality) of the decomposition is usually small, such as r=1 in the original paper.
- **Scaling factor (alpha)**: Used to stabilize training and ensure that small changes in r don't disproportionately affect the model's performance.
- **No significant inference penalty**: LoRA allows efficient fine-tuning without introducing a major computational overhead during inference.
- **Other PeFT techniques**: Methods such as fine-tuning the last layer, adapter layers, and prefix tuning also exist but may involve more parameter changes or other trade-offs.

Efficient Adaptation of Quantized LLMs (QLoRA):

- **BitsAndBytes library**: QLoRA leverages the BitsAndBytes library to implement 4-bit quantization, enabling fine-tuning of large models on smaller GPUs.
- **Quantization techniques**: QLoRA introduces three main quantization techniques:
    1. **4-bit NormalFloat**: Instead of using standard 32-bit floating-point (Float32) representations, QLoRA quantizes parameters into 4 bits (Int4), greatly reducing memory usage (from 4 bytes per parameter to 0.5 bytes).
    2. **Double Quantization**: Addresses precision loss during quantization by using two stages of quantization. Instead of one global quantization constant (c) for all values, this method allows multiple buckets, each with its own c, reducing the impact of outliers. After the first quantization, the constants are quantized again (Float32 to Float8), and during inference, a two-step dequantization is performed.
    3. **Paged Optimizers**: Helps manage memory efficiently during training by moving optimizer states between GPU and CPU memory when dealing with long input sequences. This prevents memory spikes and potential crashes during training.

- **Quantization process**: The quantization constant c is calculated as:
  c=(127/absmax(X)) times Xi
  where absmax(X) is the maximum absolute value in the vector X, and the resulting values are rounded to the nearest integer within the quantized range.
- **Precision challenges**: Regular quantization can lead to significant precision loss, particularly for outlier values, as multiple values might be mapped to the same integer. Double quantization helps mitigate this by allowing finer control over precision in different regions of the value distribution.