PEFT: Parameter Efficient Fine Tuning

Parameter Efficient Fine-Tuning diverges from full fine-tuning by **updating only a small subset of parameters.** This approach either **freezes most model weights,** focuses on specific layers, or introduces new components, significantly reducing the number of trained parameters.

Types :

1. Additive Fine-tuning
2. Partial Fine-tuning
3. Re parameterized Fine-tuning
4. Hybrid Fine-tuning

The PEFT Advantage:

1. **Memory Management**: PEFT minimizes memory requirements, sometimes enabling training on a **single GPU, r**endering the process more manageable.
2. **Avoiding Catastrophic Forgetting**: By freezing most LLM weights, PEFT mitigates **catastrophic forgetting** issues prevalent in full fine-tuning.
3. **Reduced Storage Overheads**: Unlike **full fine-tuning** that results in new models for each task, **PEFT's smaller footprint lessens storage concerns** when fine-tuning for multiple tasks

Reduced computational requirements:

Parameter efficient techniques reduce the computational demands by reducing model size, eliminating redundant parameters, and lowering precision.

Faster inference:

With smaller models and reduced computational complexity, parameter efficient fine-tuning enables faster inference, making it suitable for real-time applications.

Improved deployment on resource-constrained devices:

By reducing the model size and computational requirements, parameter efficient fine-tuning enables deployment on devices with limited resources, such as mobile devices or edge devices.

Cost savings:

By minimizing the need for extensive computational resources, parameter efficient fine-tuning reduces the associated costs for training and inference.

Maintained or improved performance:

Despite the reduction in model size and computational requirements, parameter efficient fine-tuning techniques strive to maintain or even improve the performance on downstream tasks.

Parameter Efficient Fine-Tuning Techniques:

Knowledge Distillation

Knowledge distillation is a technique that involves transferring knowledge from a large, well-performing model (known as the teacher model) to a smaller model (known as the student model). The teacher model's output probabilities serve as soft targets for training the student model. This process enables the student model to benefit from the teacher model's knowledge and generalize better. Knowledge distillation not only reduces the model size but also helps regularize the student model. It has been shown to be effective in improving performance while maintaining efficiency.

Pruning

Pruning is a technique that involves removing **unnecessary weights or connections from a pre-trained model.** By identifying and eliminating **redundant or less important parameters**, the **model's size and computational**

requirements can be significantly reduced. Pruning can be performed based on different criteria, such as magnitude-based pruning or structured pruning. Magnitude-based pruning removes weights with small magnitudes, while structured pruning removes entire neurons or filters based on their importance. Pruning enables efficient fine-tuning by eliminating redundant parameters without sacrificing performance

Quantization

Quantization is a technique that reduces the **precision of model parameters (weights) to lower memory and computational requirements**. In traditional deep learning models, parameters are typically stored as 32-bit floating-point numbers. However, quantization allows for representing these parameters with lower bit precision, such as 8-bit integers. This reduction in precision significantly reduces the memory footprint and speeds up computations. Quantization can be performed through post-training quantization, where the pre-trained model is quantized after training, or through quantization-aware training, where the model is trained with quantization in mind. Both approaches provide efficient fine-tuning options while preserving model performance.

Low-Rank Factorization

Low-rank factorization is a technique that approximates **the weight matrices of a pre-trained model with low-rank matrices.** By decomposing weight matrices into low-rank components, the number of parameters and the computational complexity of the model are reduced. Low-rank factorization methods, such as singular value decomposition (SVD) and tensor factorization, aim to find low-rank approximations that retain the most critical information for the task at hand. This technique offers parameter-efficient fine-tuning options while achieving reasonable performance.

Low-rank adaption (LoRA) is a technique to approximate the update to the linear layers in a LLM with a low-rank matrix factorization. This significantly reduces the number of trainable parameters and speeds up training with little impact on the final performance of the model.

Knowledge Injection

Knowledge injection is a technique that **enhances the performance of a pre-trained model on a specific task by injecting task-specific information**

without modifying the original parameters. This is achieved by adding task-specific modules or modifying the existing architecture to capture task-specific characteristics. Knowledge injection allows for efficient fine-tuning as it minimizes the need for extensive retraining while leveraging the knowledge already present in the pre-trained model. It is particularly useful in scenarios where minimal modification to the original model is desired.

Adapter Modules

Adapter modules are l**ightweight modules added to pre-trained models for specific tasks without modifying the original parameters.** These modules are task-specific and can be inserted into different layers of the pre-trained model's architecture. Adapter modules allow for efficient fine-tuning by enabling the model to learn task-specific information while keeping the majority of the pre-trained parameters intact. This approach provides flexibility and efficiency, as adapters can be easily added or removed for different tasks without retraining the entire model.

LORA:

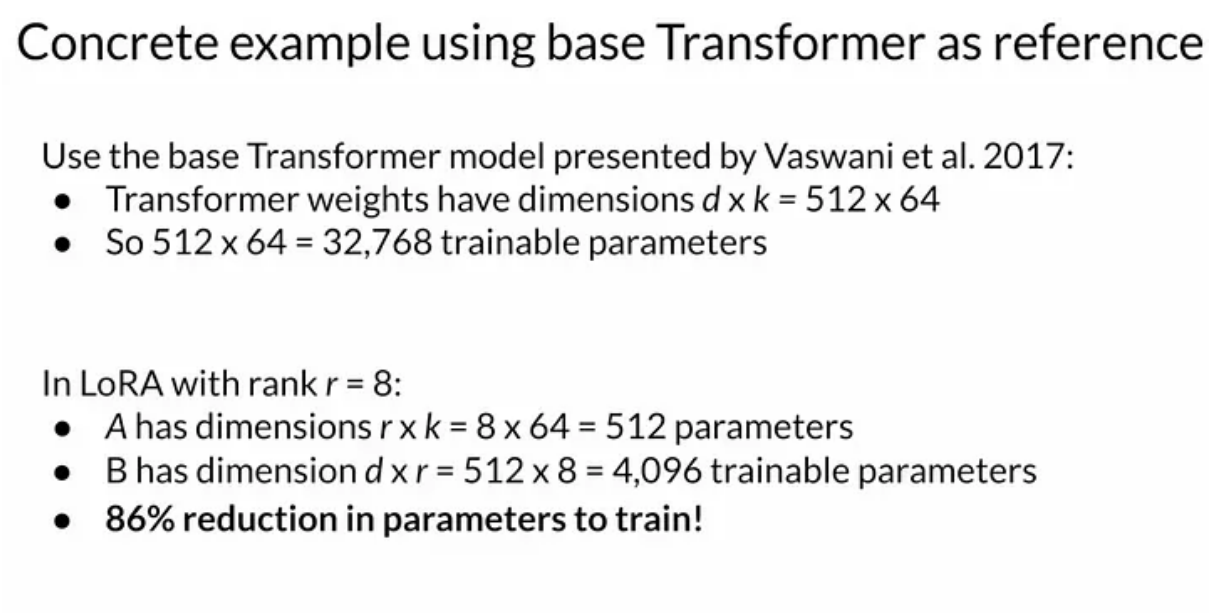
LoRA employs a strategic approach to fine-tuning by freezing all original model parameters and introducing a pair of rank decomposition matrices alongside them. These matrices are designed to have smaller dimensions, their product forming a matrix equivalent to the weights they modify. By freezing the original weights and training these smaller matrices, LoRA significantly reduces the number of trainable parameters.

The LoRA Process

1. **Training**: During fine-tuning, t**he two low-rank matrices are trained** using the same supervised learning process. This reduction in trainable parameters is particularly impactful for self-attention layers.
2. **Inference**: For inference**, the low-rank matrices are multiplied**, added to the original frozen weights, and then used to replace the original weights in the model. This results in a LoRA fine-tuned model with minimal impact on inference latency.

Inference is the process that a trained machine learning model* uses to draw conclusions from brand-new data. An AI model capable of making inferences can do so without examples of the

desired result.



Prompt Engineering

Prompt engineering is the art of crafting effective prompts that guide the LLM towards generating the desired output. It involves understanding the capabilities and limitations of the LLM, identifying the task-specific requirements, and designing prompts that align with both

Here's a step-by-step guide to prompt engineering:

1. **Identify the task:** Clearly define the task you want the LLM to perform, such as answering questions, or summarizing information.
2. **Analyze the LLM:** Research and understand the capabilities and limitations of the LLM you will be using. This includes strengths, weakness, and any

specific guidelines or requirements.

3. **Design the prompt:** Craft a prompt that clearly communicates the task to the LLM. The prompt should be concise, specific, and provide the necessary context and instructions.
4. **Provide examples and demonstrations:** If possible, provide the LLM with examples of desired outputs or demonstrations of how the task should be performed. This helps the LLM better understanding of your expectations.
5. **Iterate and refine:** Test different prompts and analyze the results. Iterate on the prompt, refining it based on the feedback and performance of the LLM.

Prompt tuning :

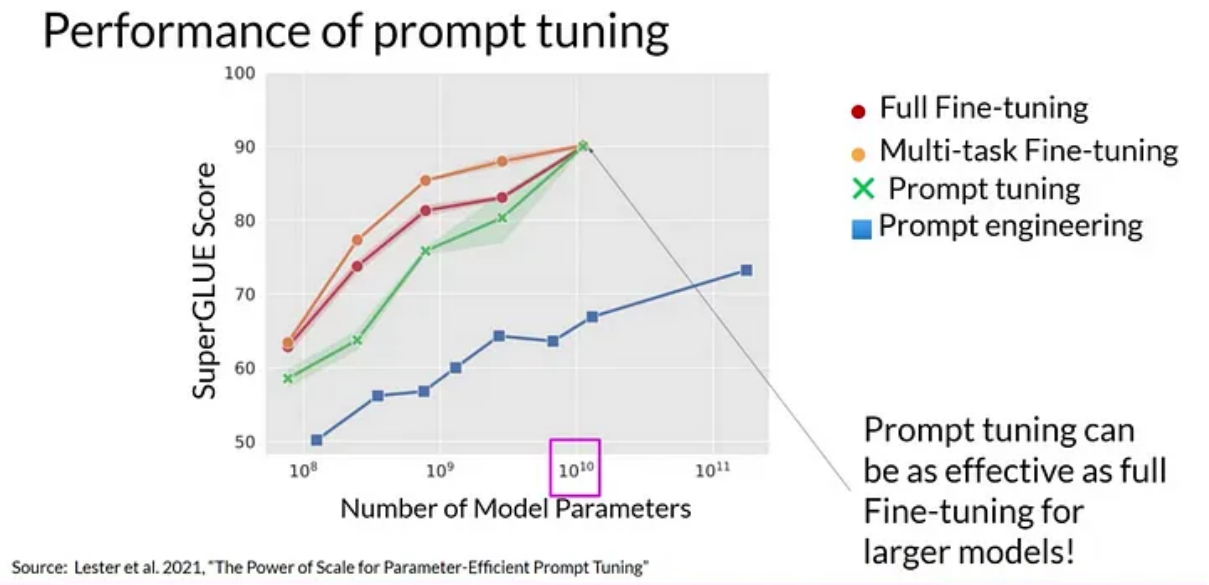
Prompt Tuning distinguishes itself from prompt engineering by introducing a novel approach to parameter **optimization.** Unlike conventional methods that require manual effort and are constrained by context window length, **Prompt Tuning involves adding trainable tokens, termed soft prompts**, to the input prompt. These virtual tokens, existing in a continuous embedding space, allow the model to learn optimal values through supervised learning. With only a few parameters being trained, Prompt Tuning proves to be highly efficient, particularly as the model size increases.

prompt tuning process:

1. Define the task: Similar to prompt engineering, the first step is to clearly define the task you want the LLM to perform.
2. Initialize the prompt: Start with a baseline prompt that captures the essence of the task. This initial prompt can be generated manually or with the help of the prompt engineering techniques.
3. Choose an optimization method: Select an appropriate optimization method, such as Bayesian optimization or gradient-based optimization, to fine-

tuning the prompt parameters.

4. Fine-tune the prompt: Use the optimization method to iteratively adjust the prompt parameters and evaluate the performance of the LLM on the task. The goal is to find the prompt that maximize the LLM's performance.
5. Evaluate and refine: Continuously evaluate the performance of the tuned prompt and make adjustments as needed. The prompt tuning process can be iterative, with multiple rounds of optimization and refineme



QLORA:

Quantized Low-Rank Adaptation (QLoRA) aims to efficiently fine-tune massive models (like a 65B parameter model) on limited GPU memory without compromising the model's performance. It builds on LoRA's principles and introduces 4-bit NormalFloat (NF4) quantization and Double Quantization techniques. In other words, QLoRA is an advanced technique designed for parameter-efficient fine-tuning of large pre-trained language models (LLMs). It builds upon the principles of Low-Rank Adaptation (LoRA) but introduces additional quantization to enhance parameter efficiency further

QLoRA leverages a frozen, 4-bit quantized pretrained language model and backpropagates the gradients into Low Rank Adapters (LoRA). This combination seems to optimize both computation (by using low-bit quantization) and the number of parameters (using low-rank structures).

1. **Low-Rank Adaptation:** Like LoRA, QLoRA injects trainable low-rank matrices into the architecture (specifically, the Transformer layers) of pretrained LLMs. This technique ensures efficient fine-tuning by optimizing these low-rank matrices rather than the full model, resulting in fewer trainable parameters and reduced computational costs.
2. **Quantization:** The standout feature of QLoRA is its use of quantization to achieve higher memory efficiency.

- NF4 Quantization: QLoRA employs 4-bit NormalFloat (NF4) quantization. By transforming all weights to a specific distribution that fits within the range of NF4, this technique can efficiently quantify weights without needing intricate algorithms for quantile estimation.

Goal:

- QLoRA minimizes memory utilization during fine-tuning of large language models (LLM) without **compromising on performance** as compared to the conventional 16-bit model fine-tuning.
- **Mechanism:**
- **4-bit Quantization:** Pretrained language models are compressed using 4-bit quantization.
- **Low-Rank Adapters**: After freezing the quantized parameters of the language model, Low-Rank Adapters (LoRA) are added as trainable parameters.
- **Backpropagation:** Gradients are backpropagated through the frozen**, quantized pretrained model**, specifically targeting the LoRA layers, which

are the only trainable parameters during fine-tuning.

- **Data Types:**
- Storage: QLoRA employs a storage data type, typicall**y 4-bit NormalFloat,** for the base model weights.
- Computation: A 16-bit BrainFloat data type is utilized for computations. Weights are dequantized to this computation data type for forward and backward passes. Weight gradients are only computed for the LoRA parameters.
- **Results:**
- Comparable performance with 16-bit fine-tuning methods in various tests.
- Models fine-tuned with QLoRA, specifically the Guanaco models trained on the OpenAssistant dataset, deliver state-of-the-art chatbot performance, closely matching that of ChatGPT on the Vicuna benchmark.

- **Advantages:**
- **Further Memory Reduction:** QLoRA achieves higher memory efficiency through quantization.
- **Preserved Performance:** Despite being more parameter-efficient, QLoRA ensures high model quality, on par or even superior to fully fine-tuned models in various tasks.
- **Applicability:** QLoRA is versatile, being compatible with various LLMs like RoBERTa, DeBERTa, GPT-2, and GPT-3.
- The `transformers` library, maintained by Hugging Face, offers an integration of the QLoRA method. This integration facilitates the efficient quantization of supported models.