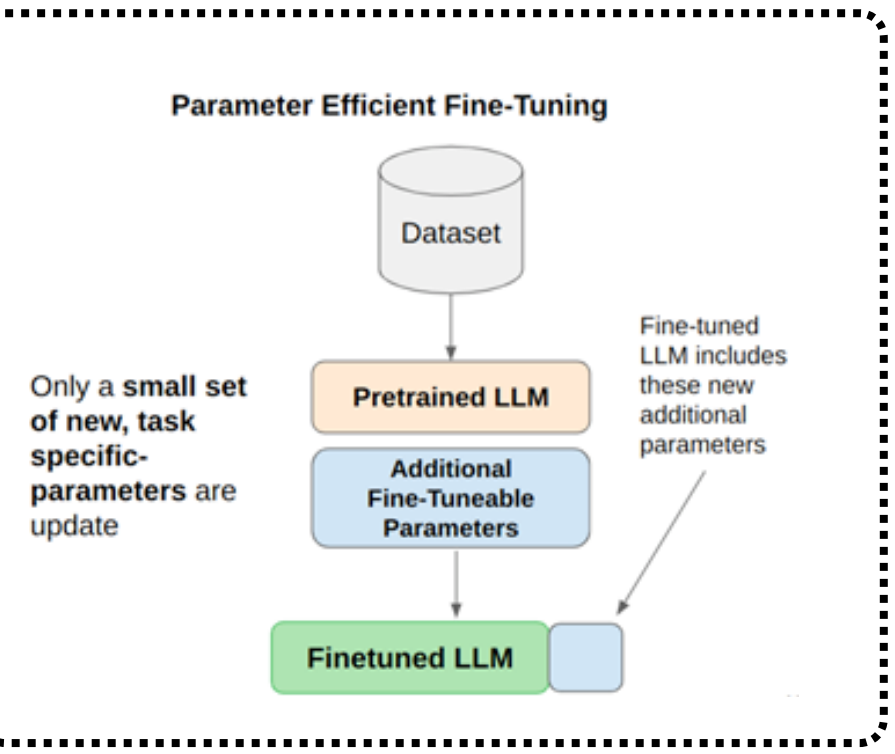
Mastering LLMs

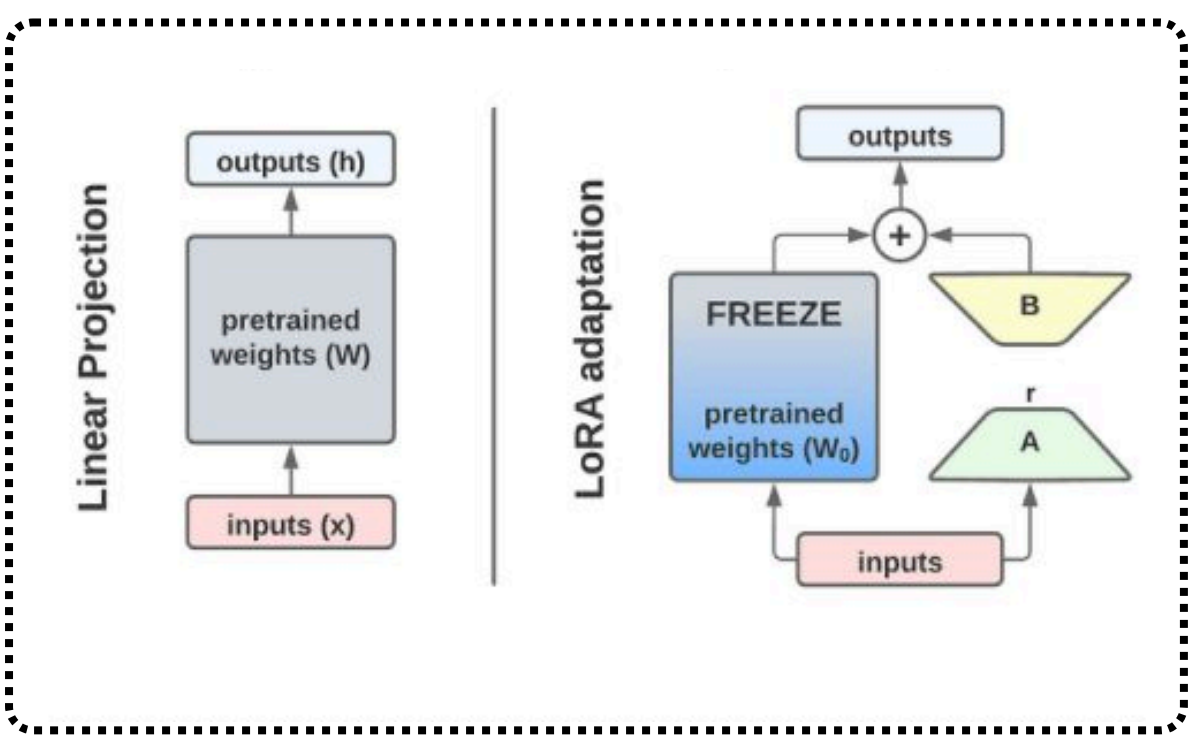
Analytics Vidhya

Day 23: Boosting Efficiency Techniques in LLMs

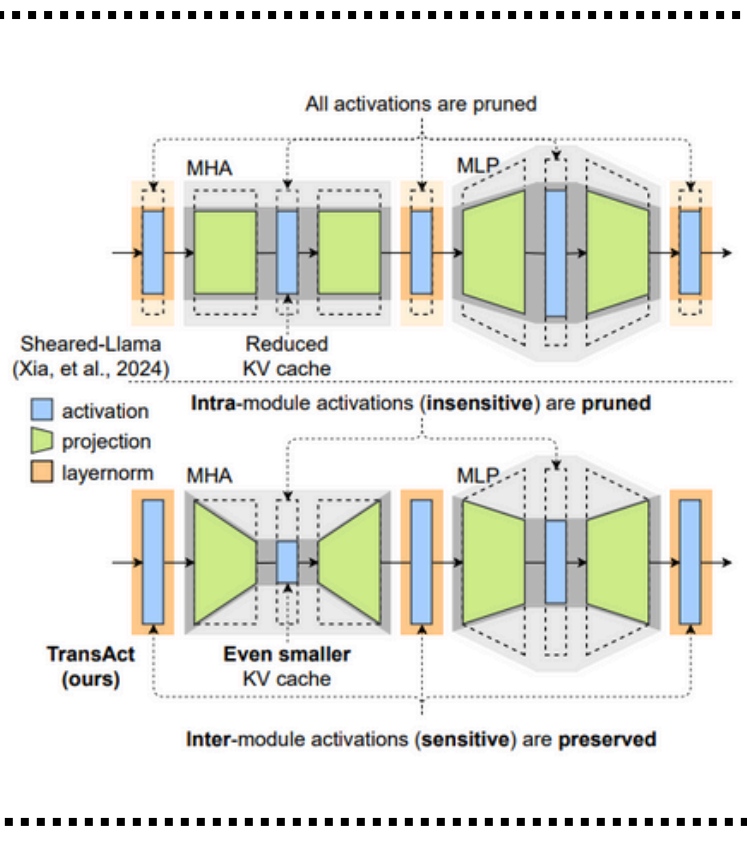
Parameter Efficient Fine-Tuning (PEFT)



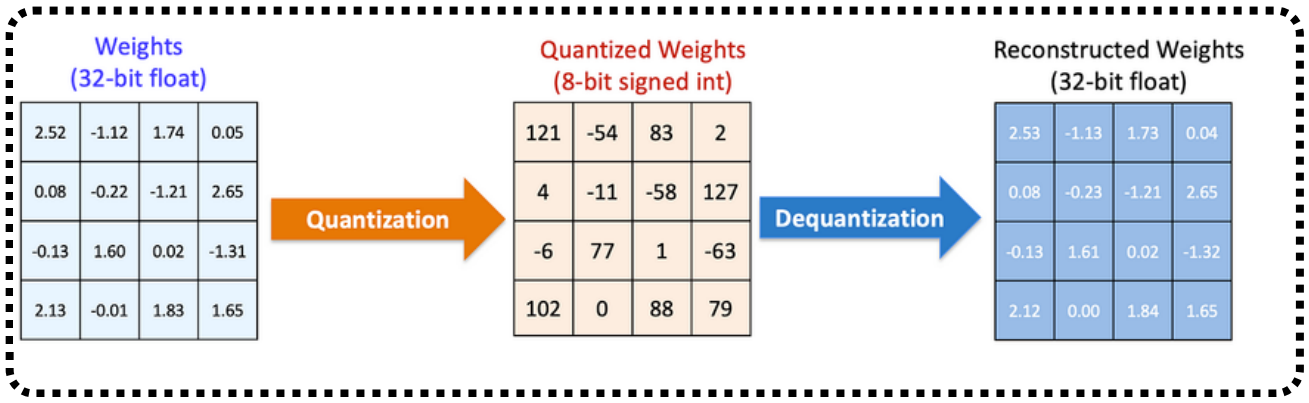
Low-Rank Adaptation(LoRA)



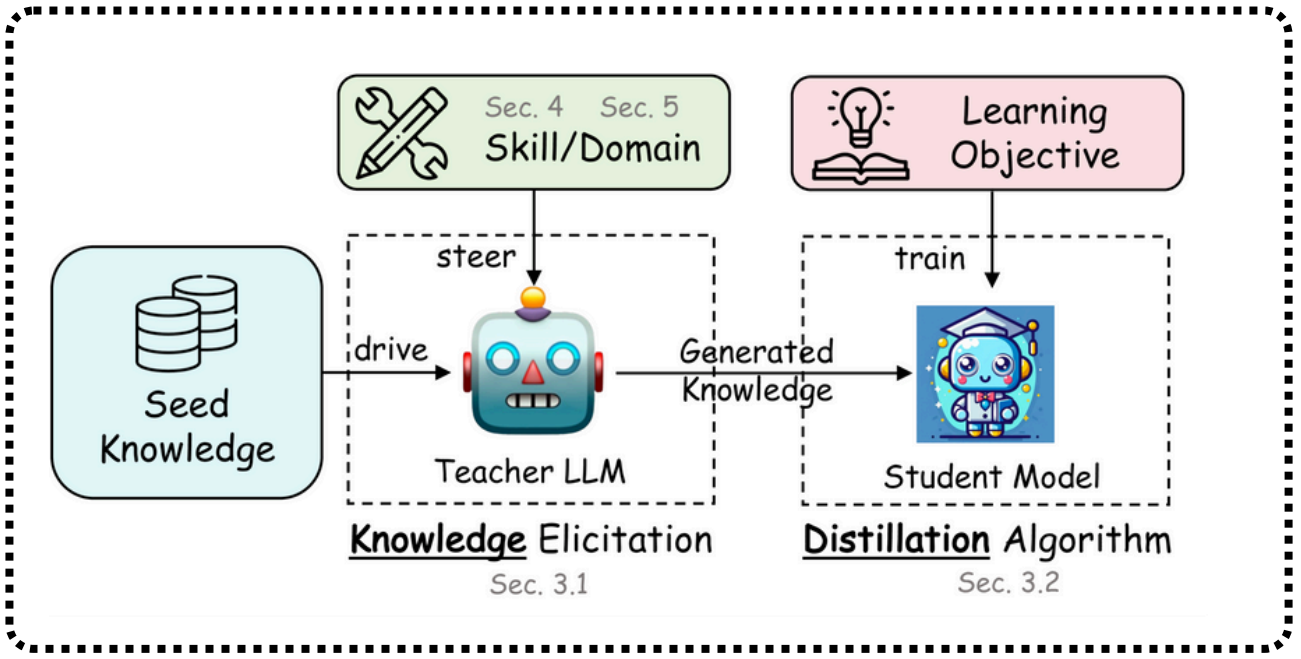
Quantization



Model Pruning



Knowledge Distillation



Boosting efficiency in Large Language Models (LLMs) is critical to make them more computationally feasible, particularly as their size grows exponentially. These models, while powerful, are resource-intensive and require significant computational resources to train, fine-tune, and deploy. To overcome these challenges, researchers have developed a variety of techniques designed to improve both computational efficiency and resource consumption without sacrificing performance.

Below are some of the most important techniques used to boost efficiency in LLMs:

1. Parameter Efficient Fine-Tuning (PEFT)
2. Low-Rank Adaptation (LoRA)
3. Knowledge Distillation
4. Model Pruning
5. Quantization
6. Efficient Transformers
7. Mixed Precision Training
8. Parallelism Techniques
9. Sparse Attention Mechanisms

PEFT

PEFT techniques allow large models to be fine-tuned on specific tasks without updating all parameters in the model. This reduces the computational cost and memory requirements by only modifying a small subset of model parameters.

- **Adapters**: Small trainable layers added between the pre-existing layers of a large model. These layers are fine-tuned, while the original weights of the model remain fixed. Adapters can be added to multiple layers or just a subset, depending on the task.

- **Prompt Tuning**: Instead of tuning the entire model, the input prompt (the part that provides context to the model) is fine-tuned. This is especially useful for language models that use prompt-based architectures (like GPT or T5).

Low-Rank Decomposition: This technique decomposes the large weight matrices of a neural network into low-rank approximations. By updating only the low-rank matrices, the number of parameters that need to be adjusted is drastically reduced, leading to lower memory usage and faster fine-tuning.

Low-Rank Adaptation (LoRA)

LoRA focuses on using low-rank matrix approximations to fine-tune only a subset of parameters in LLMs, making adaptation more efficient and memory-friendly. Instead of modifying the entire weight matrix, LoRA introduces small low-rank matrices that capture only the essential changes needed for fine-tuning.

How it Works

In a standard neural network, the weight matrix WWW is updated directly during fine-tuning:

$$W' = W + \Delta W$$

where ΔW is the modification learned during training.

Instead of learning a full-rank ΔW, LoRA **decomposes** it into two smaller matrices:

$$\Delta W = AB$$

where:

- $A \in \mathbb{R}^{m \times k}$
- $B \in \mathbb{R}^{k \times n}$
- $k \ll \min(m, n)$, meaning that A and B have **far fewer parameters** than W.

Thus, the updated weight matrix becomes:

$$W' = W + AB$$

Since **only** A and B are trained while keeping W frozen, **LoRA significantly reduces computational cost and memory usage**, making fine-tuning **faster and more efficient** without modifying the full model.

Knowledge Distillation

Knowledge distillation is a technique where a smaller model (the "student") learns to replicate the behavior of a larger, more complex model (the "teacher"). The goal is to create a more efficient version of a large model without sacrificing its performance.

- **Process**: The large pre-trained model (teacher) is used to generate predictions on a given task. These predictions, known as "soft labels," are then used to train the smaller student model. The student model learns not just the final prediction, but also the intermediate representations used by the teacher.

- **Advantages**: The distilled model is typically much smaller and faster, while maintaining much of the performance of the large model.

Model Pruning

Pruning involves removing redundant or unimportant weights in the neural network, thereby reducing the model's size and improving efficiency.

- **How it Works**: Weights that contribute little to the model's performance are identified (using methods like weight magnitude or gradient-based pruning) and are removed, reducing the number of parameters. This leads to a smaller and faster model.

- **Types of Pruning**:

    - **Magnitude-based pruning**: Removing weights with the smallest absolute values.

    - **Gradient-based pruning**: Removing weights with the smallest gradients, assuming these weights are less important for learning.

- **Impact**: Pruned models are more efficient in terms of memory and computation, but excessive pruning can lead to a drop in model performance.

Quantization

Quantization reduces the precision of the model's weights, converting them from floating-point numbers (e.g., 32-bit) to lower precision formats (e.g., 16-bit, 8-bit). This reduces memory usage and speeds up computation without drastically affecting model accuracy.

- **How it Works**: Quantization involves mapping the high-precision weights (32-bit or 16-bit) to a lower bit width (like 8-bit or even binary), reducing the model size and speeding up both inference and training.

- **Advantages**:

    - Reduced memory footprint
    - Faster inference times, especially in hardware that supports lower precision
    - Improved energy efficiency

- **Challenges**: Fine-tuning and maintaining performance can be tricky, as reducing precision might cause a drop in accuracy if not handled carefully.

Efficient Transformers

Transformers, the backbone architecture for many state-of-the-art LLMs (like GPT and BERT), have a computational complexity of O(n^2), where n is the sequence length. As sequences become longer, this complexity becomes prohibitive. Efficient transformer variants aim to reduce this complexity.

- **Reformer**: Uses locality-sensitive hashing (LSH) to reduce the time complexity of attention from O(n^2) to O(nlog n).

- **Linformer**: Reduces the full attention mechanism to a low-rank approximation, thereby speeding up computation.

- **Longformer**: Uses a sliding window attention mechanism, reducing the complexity of the attention mechanism from O(n^2) to O(n).

- **Performer**: Replaces the softmax function in attention with a kernel approximation, speeding up computation for large-scale inputs.

Mixed Precision Training

In mixed precision training, both 16-bit and 32-bit floating-point precision are used during training. This allows for reduced memory usage and faster computations, while still maintaining the model's accuracy.

- **How it Works**: The model’s weights are stored in 32-bit precision, but computations (like gradients) are done in 16-bit precision. This reduces the memory footprint during training and speeds up the process, particularly on GPUs that support mixed-precision operations.

- **Benefits**:
  - Faster training times
  - Lower memory usage
  - Can be combined with other techniques like quantization for further optimization

Parallelism Techniques

When working with large models, parallelism can be used to distribute the computation across multiple processors or devices. Several parallelism strategies include:

- Data Parallelism: Splits the dataset into smaller batches, which are processed in parallel across multiple devices. Each device holds a replica of the model.

- Model Parallelism: Splits the model itself across multiple devices, allowing the model to be too large to fit on a single device but still enabling training.

- Pipeline Parallelism: Breaks the model into stages and processes them in parallel, with each stage operating on different data.

Parallelism helps reduce training time by distributing the workload, especially when the model is too large to fit into a single machine.

Sparse Attention Mechanisms

In large models, full attention mechanisms can become computationally expensive. Sparse attention reduces the number of attention heads or connections in the attention matrix by making most of them zero (sparse).

- **How it Works**: Attention is computed only over a subset of tokens rather than all tokens, often based on predefined patterns or learned sparse patterns.

- **Variants**:

    - **Linformer**: Implements low-rank attention.

    - **BigBird**: Uses sparse attention to limit interactions between tokens, which can speed up attention calculation while preserving performance.

Analytics Vidhya

Stay Tuned for **Day 24** of

Mastering LLMs