

Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica



Lesson 17 Fine Tuning

Nicola Capuano and Antonio Greco

DIEM – University of Salerno



Outline

- Types of fine tuning
- Parameter Efficient Fine Tuning (PEFT)
- Instruction Fine-Tuning






Types of fine tuning

Pros and cons of full fine tuning

- Fine-tuning refers to adapting a pre-trained LLM to a specific task by training it further on a task-specific dataset.
- Why Fine-Tune?
 - Specialize LLMs for domain-specific tasks.
 - Improve accuracy and relevance for specific applications.
 - Optimize performance on small, focused datasets.
- Full Fine-Tuning, which updates all model parameters, allows to achieve high accuracy for specific tasks by fully leveraging the model's capacity, but it is computationally expensive and risks to overfit on small datasets.

Other types of fine tuning

- **Parameter-Efficient Fine-Tuning (PEFT):** Updates only a subset of the parameters.
 - Examples: LoRA, Adapters.
- **Instruction Fine-Tuning:** Used to align models with task instructions or prompts (user queries) enhancing usability in real-world applications.
- **Reinforcement Learning from Human Feedback (RLHF):** Combines supervised learning with reinforcement learning, rewarding models when they generate user-aligned outputs.



Parameter efficient fine tuning (PEFT)

Parameter-Efficient Fine-Tuning

- **Parameter-Efficient Fine-Tuning (PEFT)** is a strategy developed to fine-tune large-scale pre-trained models, such as LLMs, in a computationally efficient manner while requiring fewer learnable parameters compared to standard fine-tuning methods.
- PEFT methods are especially important in the context of LLMs due to their massive size, which makes full fine-tuning computationally expensive and storage-intensive.
- PEFT is ideal for resource-constrained settings like edge devices or applications with frequent model updates.
- These techniques are supported and implemented in Hugging Face transformers and, in particular, in **peft** library.

PEFT techniques

- **Low-Rank Adaptation (LoRA):** Approximates weight updates by learning low-rank matrices, performing a small parameterized update of the weight matrices in the LLM. It is highly parameter-efficient and widely adopted for adapting LLMs.
- **Adapters:** They are small and trainable modules inserted within the transformer layers of the LLM, that allow to keep the pre-trained model's original weights frozen.
- **Prefix Tuning:** Learns a set of continuous task-specific prefix vectors for attention layers, keeping the original model parameters frozen.

Low-Rank Adaptation (LoRA)

- LoRA assumes that the changes required to adapt a pre-trained model for a new task lie in a low-dimensional subspace.
- Instead of fine-tuning all the parameters of the model, LoRA modifies only a small, trainable set of low-rank matrices that approximate these task-specific changes.

1. Base Model: A pre-trained transformer model is represented by its weight matrices W

2. Low-Rank Decomposition: Instead of directly modifying W , LoRA decomposes the weight update into two low-rank matrices:

$$\Delta W = A \times B$$

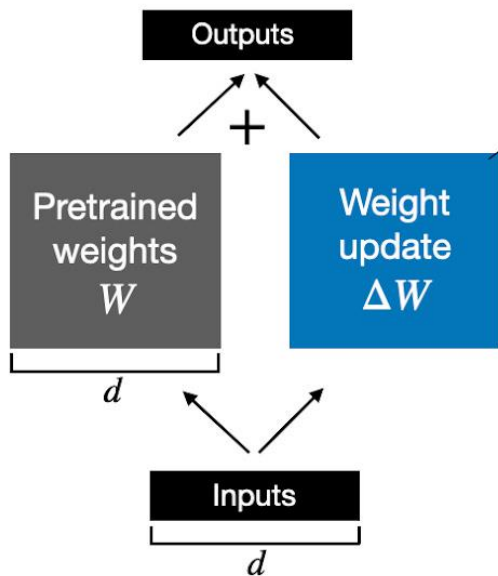
- A is a low-rank matrix ($m \times r$)
- B is another low-rank matrix ($r \times n$)
- r is the rank, which is much smaller than m or n , making A and B parameter-efficient.

3. Weight Update: The effective weight during fine-tuning becomes

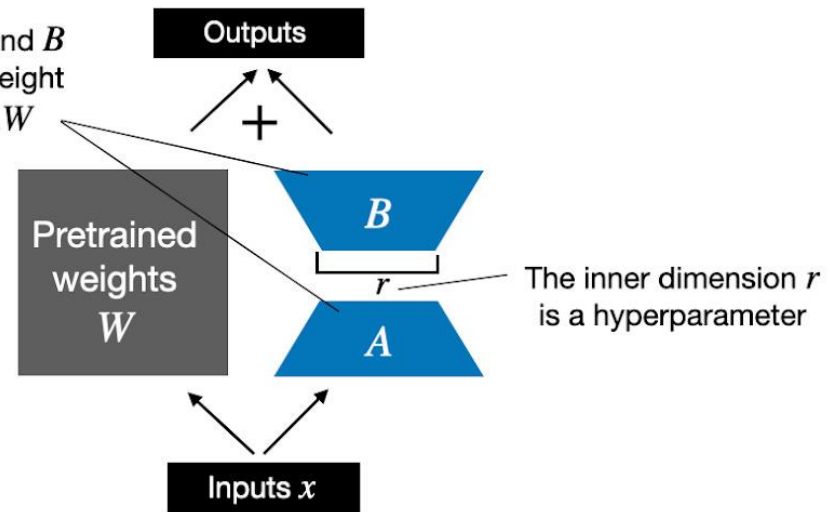
$$W' = W + \Delta W = W + A \times B$$

Low-Rank Adaptation (LoRA)

Weight update in **regular finetuning**



Weight update in **LoRA**

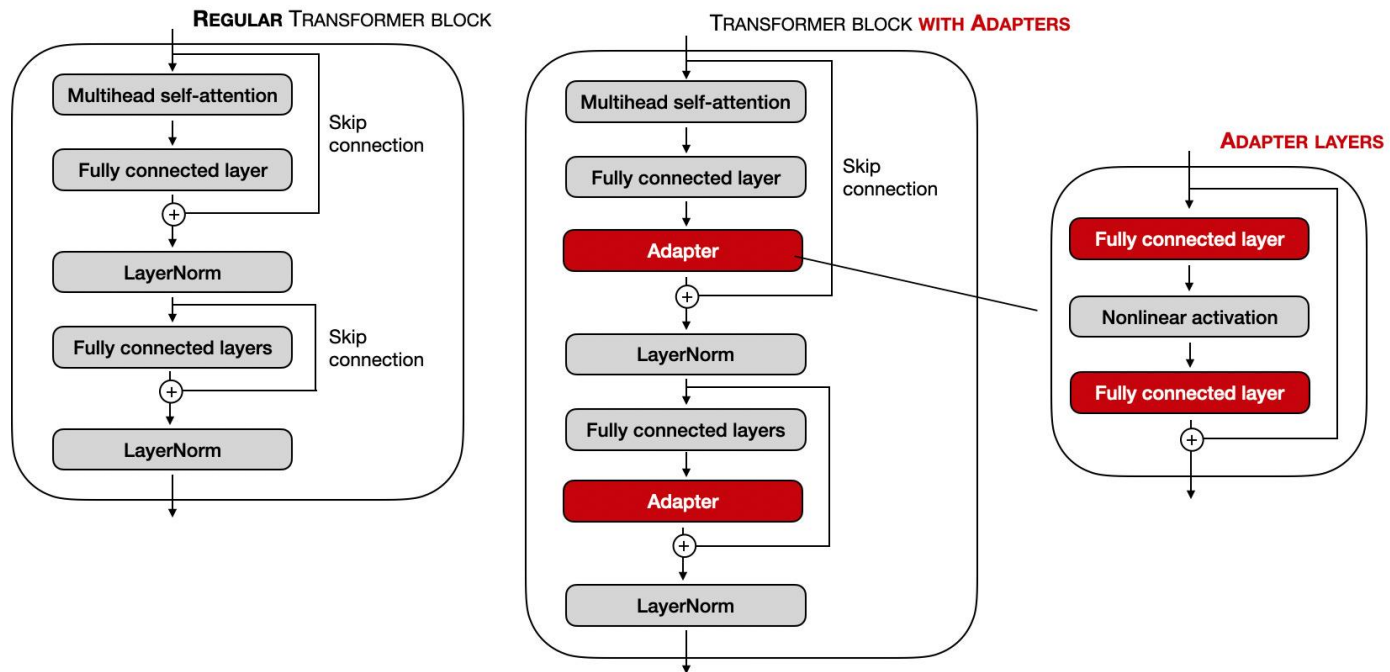


How LoRA works

- **Freezing Pre-Trained Weights:** LoRA keeps the original weight matrices W of the LLM frozen during fine-tuning. Only the parameters in A and B are optimized for the new task (the pre-trained knowledge is preserved).
- **Injecting Task-Specific Knowledge:** The low-rank decomposition $A \times B$ introduces minimal additional parameters (less than 1% of the original model parameters) while allowing the model to learn task-specific representations.
- **Efficiency:** The number of trainable parameters is proportional to $r \times (m+n)$, which is significantly smaller than the full $m \times n$ weight matrix.
- **Inference Compatibility:** During inference, the modified weights $W' = W + A \times B$ can be directly used, making LoRA-compatible models efficient for deployment.

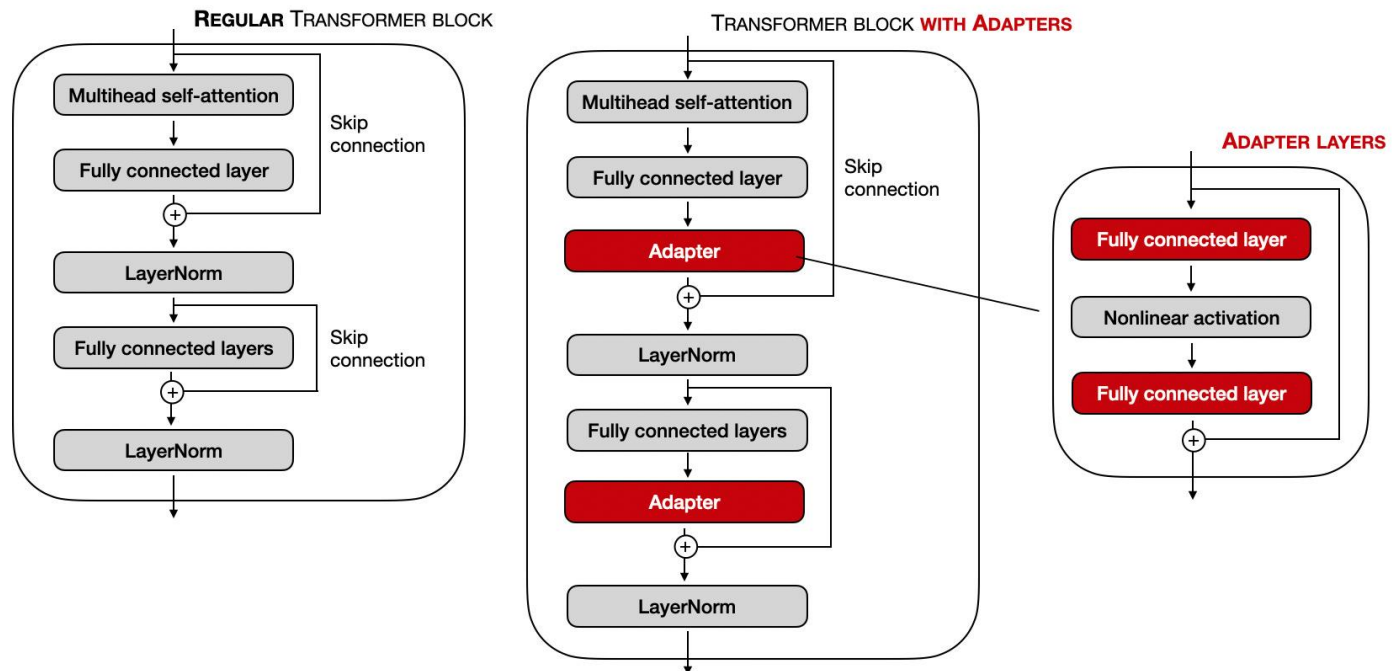
Adapters

- Adapters are lightweight, task-specific neural modules inserted between the layers of a pre-trained transformer block.
- These modules are trainable, while the original pre-trained model parameters remain frozen during fine-tuning.



Adapters

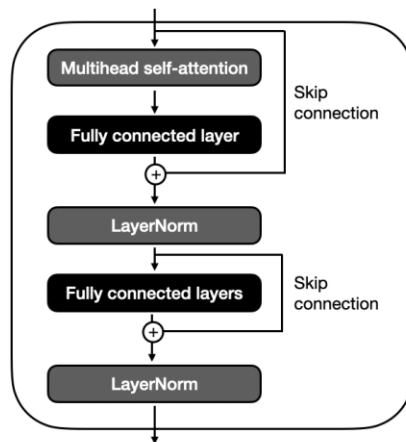
- Adapters require training only the small fully connected layers, resulting in significantly fewer parameters compared to full fine-tuning.
- Since the base model remains frozen, the general-purpose knowledge learned during pre-training is preserved.



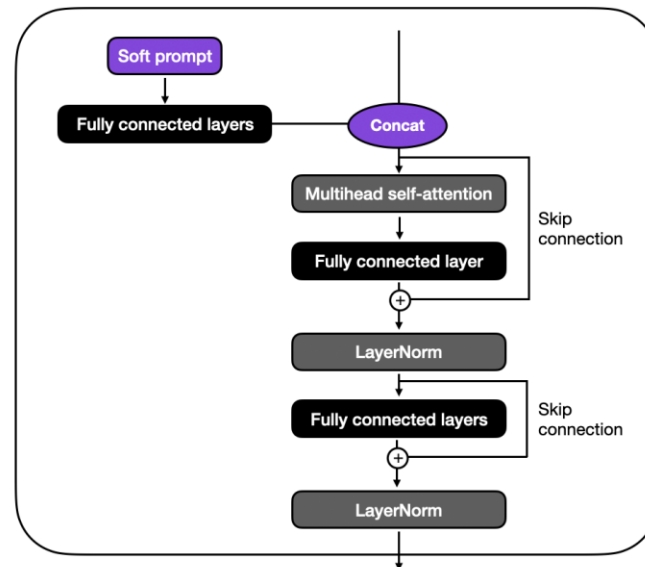
Prefix tuning

- Instead of modifying the LLM's internal weights, prefix tuning introduces and optimizes a sequence of trainable "prefix" tokens prepended to the input.
- These prefixes guide the LLM's attention and output generation, enabling task-specific adaptations with minimal computational and storage overhead.

REGULAR TRANSFORMER BLOCK



TRANSFORMER BLOCK WITH PREFIX



Prefix tuning

- The input sequence is augmented with a sequence of prefix embeddings:

Modified Input: [Prefix]+[Input Tokens]

- Prefix: A sequence of m trainable vectors of size d , where d is the model's embedding dimensionality.
 - Input Tokens: The original token embeddings from the input sequence.
- Prefix embeddings influence attention by being prepended to the keys (K) and values (V), conditioning how the model attends to the input tokens.
- Only the prefix embeddings are optimized during fine-tuning. Backpropagation updates the prefix parameters to align the model's outputs with task-specific requirements.
- m controls the trade-off between task-specific expressiveness and parameter efficiency. Longer prefixes can model more complex task-specific conditioning but may increase memory usage.



Instruction fine tuning

Instruction fine tuning

- **Instruction fine-tuning** is a specialized approach for adapting large language models (LLMs) to better understand and respond to user instructions.
- This fine-tuning process involves training the model on a curated dataset of task-specific prompts paired with corresponding outputs.
- The objective is to improve the model's ability to generalize across a wide variety of instructions, enhancing its usability and accuracy in real-world applications.
- By training on human-like instructions, the model becomes more aligned with user expectations and natural language queries.

How instruction fine tuning works

- A diverse set of instructions and outputs is compiled. Each example consists of:
 - Instruction: A clear, human-readable prompt (e.g., "Summarize the following text in one sentence").
 - Context (optional): Background information or data required to complete the task.
 - Output: The expected response to the instruction.
- The LLM, pre-trained on a general corpus, is fine-tuned using the instruction-response pairs. During training, the model learns to:
 - Recognize the intent of the instruction.
 - Generate outputs that are coherent, accurate, and contextually appropriate.
- The dataset may include examples from various domains and task types. This diversity ensures the model can generalize beyond the specific examples it has seen during fine-tuning.

Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica



Lesson 17 Fine Tuning

Nicola Capuano and Antonio Greco

DIEM – University of Salerno

