LLMOps

Session 3: Fine-tuning and Evaluation

Albert School 2025-2026

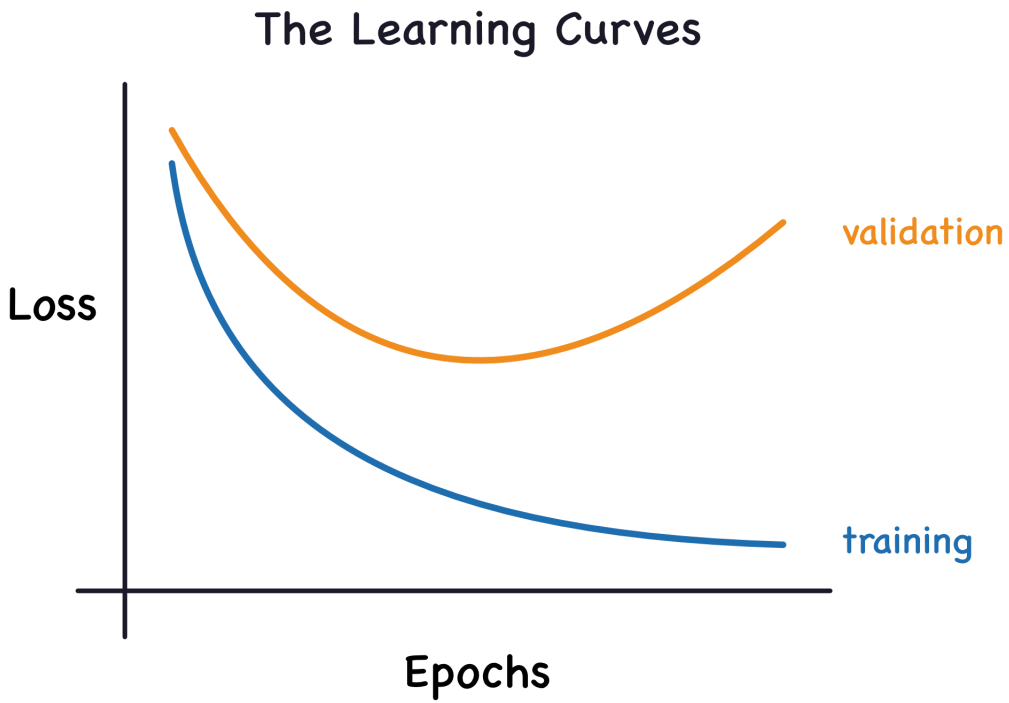
Mathieu SOUL

Lesson Plan

1. What do you need for a good fine-tuning?
2. Fine-tuning with LoRA
3. Evaluating LLMs

What do you need for a good fine-tuning?

What do you need for a good fine-tuning?



Where should training stop here?

To be able to evaluate if your fine-tuning is going well, you need to be sure your model is **fitting**, but **not over-fitting**!

For this you need a **validation dataset**, and regular evaluation along the epochs that can be visualized.

Remember to **log key hyper-parameters** to be able to reproduce your results!

Fine-tuning with LoRA

1. Quantization
2. LoRA Concept
3. LoRA Hyper-parameters

Quantization

Quantization is a technique that **reduces the precision** of the numbers used to represent a model's parameters, typically from 32-bit floating point to lower bit formats like 16-bit or 8-bit integers.

Appears a trade-off:

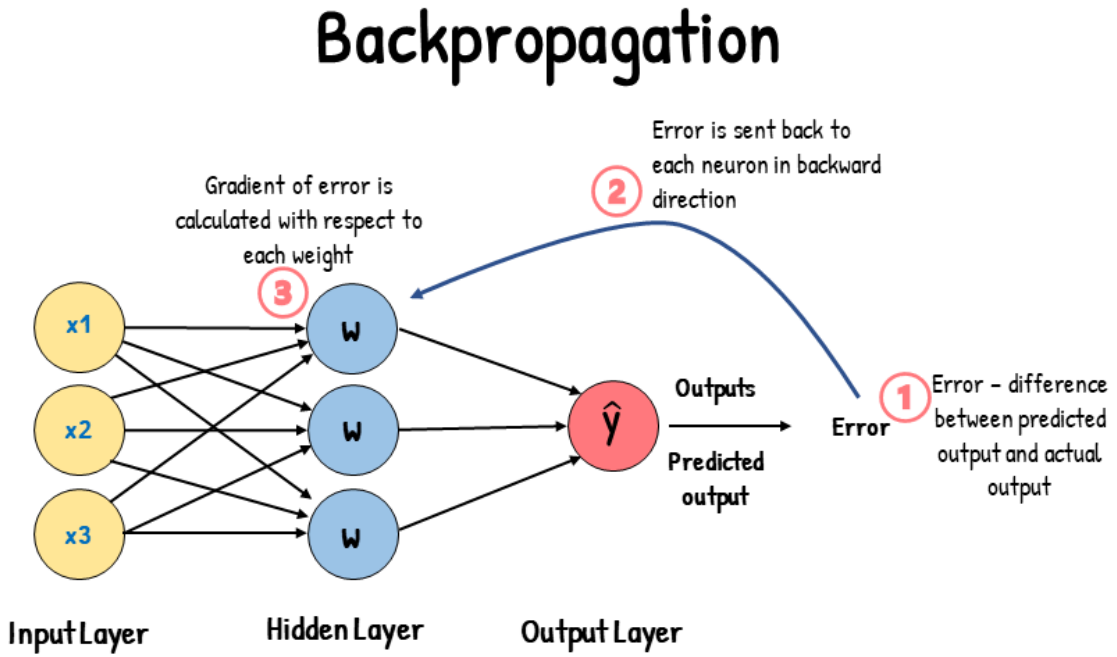
- **Pros**: Reduced memory usage, we can realistically use a single GPU
  - Faster back-propagation during fine-tuning
  - Faster inference
- **Cons**: Potential loss of model accuracy due to reduced precision.

With LLMs, **quantization is often used** because the precision loss is small compared to the benefits in fine-tuning and inference speed.

LoRA Concept - Back-propagation

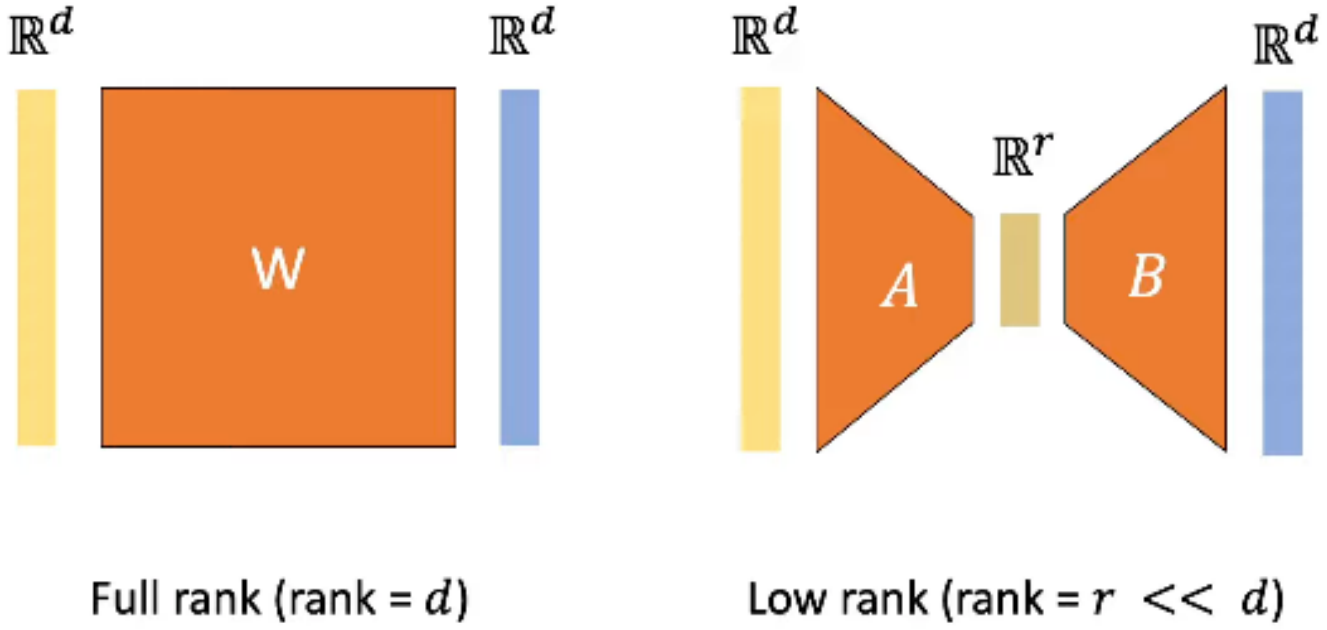
When fine-tuning a neural network, we can update all of its parameters **in place** by back-propagating the error through the entire network.

The computation cost is **proportional to the number of parameters to update!**



LoRA Concept - Intrinsic Rank Hypothesis

The key concept behind LoRA is the **Intrinsic Rank Hypothesis**: significant changes to the neural network's behavior can be captured in a **lower-dimensional** representation.

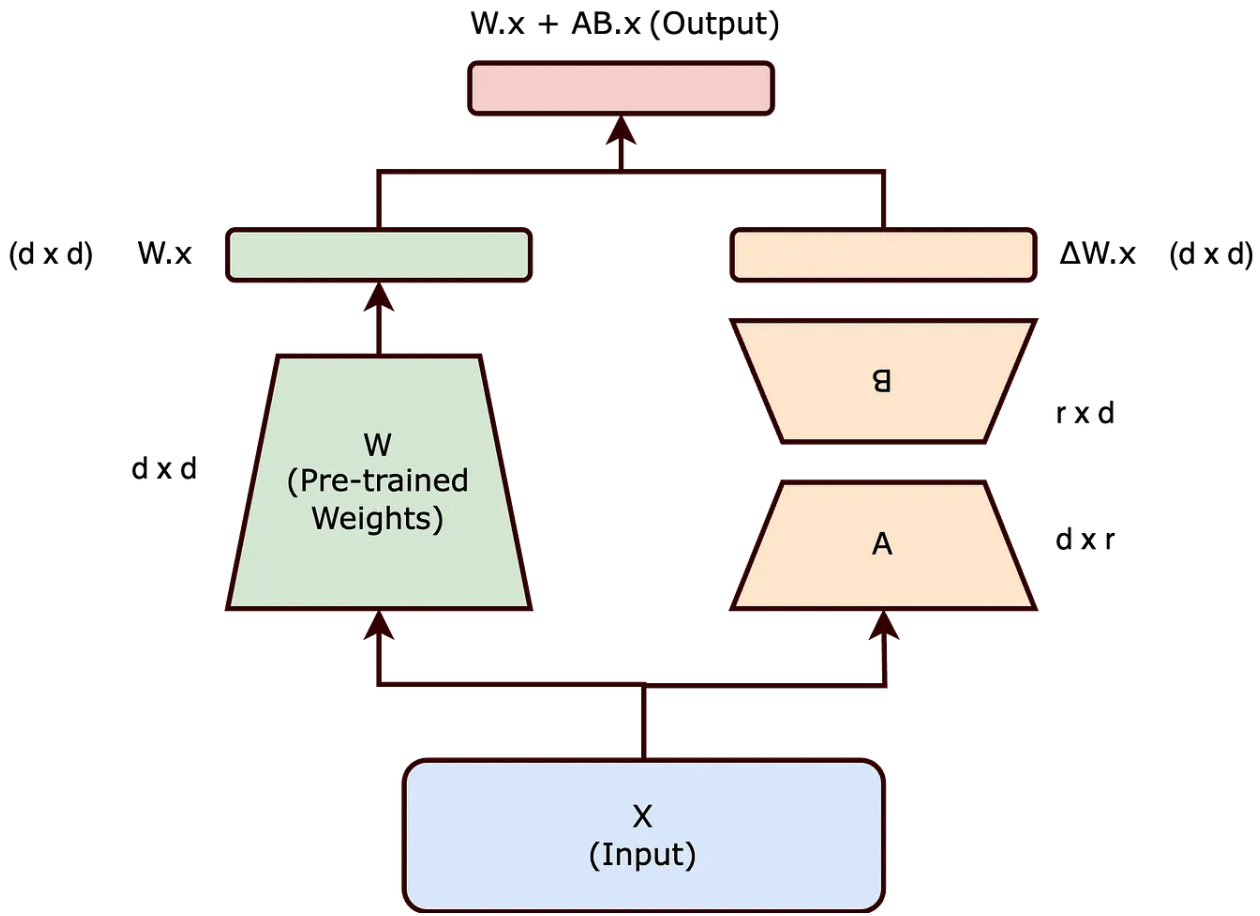


A low rank matrix has fewer parameters (d*d > 2*d*r)!

LoRA Concept - Training mechanism

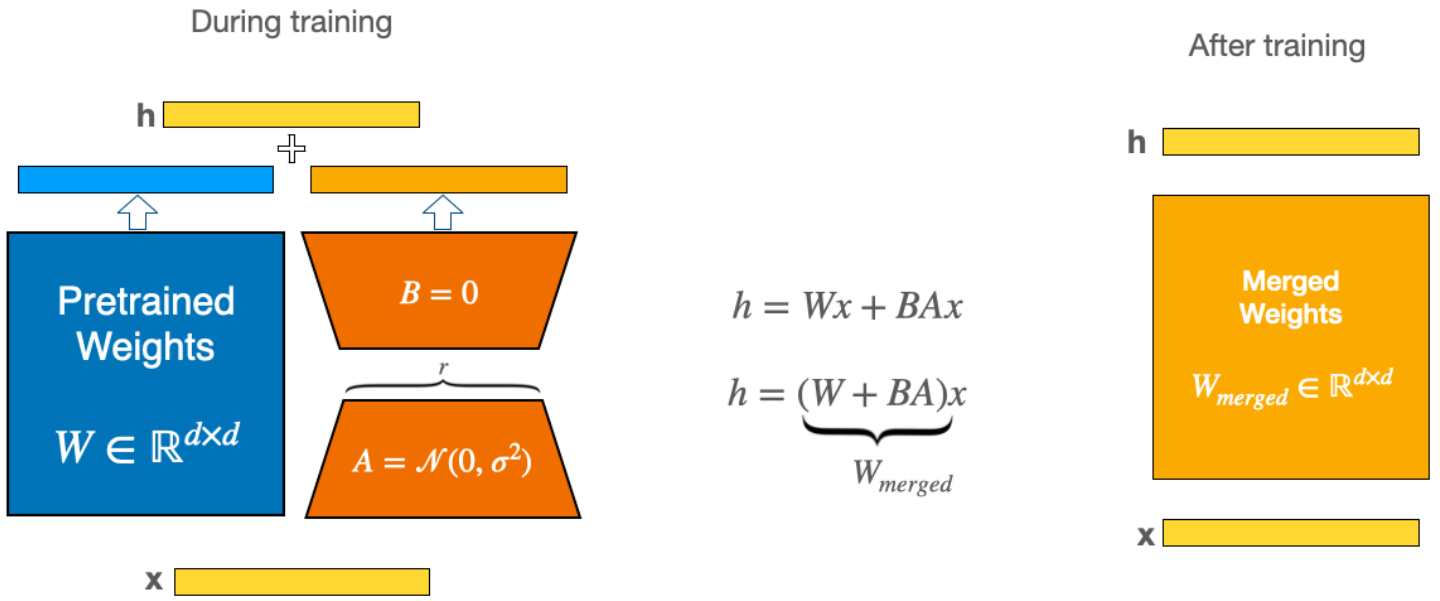
Instead of updating the full (quantized) weight matrix **W**, LoRA introduces two smaller matrices **A** and **B** such that **W' = W + BA**.

During fine-tuning, only **A** and **B** are updated by back-propagation, while **W** is **frozen**!



LoRA Concept - Inference mechanism

At inference, LoRA merges the low-rank update into the original weights: this means inference time is the same as a standard model!



The LoRA matrices being very small, the memory overhead is negligible.

LoRA Hyper-parameters

When using LoRA, you have to choose a few hyper-parameters:

- **Rank (r)**: The rank of the matrices A and B.
- **Target Modules**: The specific layers in the model where LoRA will be applied (e.g., attention layers, feed-forward layers).
- **Alpha (α)**: A scaling factor that controls the impact of the LoRA update on the original weights.
- **Task type**: The nature of the task (e.g., causal language modeling, classification)/

We'll be using Hugging Face's **PEFT** (Parameter-Efficient Fine-Tuning) library, which provides a simple interface to apply LoRA to pre-trained models.

Other generic hyper-parameters

In addition to the Lora-specific hyper-parameters, you will also need to tune the usual hyper-parameters for training, for instance:

- **Learning Rate**: Controls how much to change the model in response to the estimated error each time the model weights are updated.
- **Batch Size**: The number of training examples utilized in one iteration.
    - A larger batch size can lead to more stable gradient estimates but requires more memory.
- **Number of Epochs**: The number of complete passes through the training dataset.

Evaluating LLMs

The challenge of evaluating text generation

Evaluating a LLM that classifies or that regresses is easy: you can use standard metrics like **accuracy**, **MSE**...

Evaluating text generation is much **harder**!

```
Describe the Mona Lisa painting.
> Painting by Leonardo da Vinci, showing a woman with a mysterious smile.

Describe the Mona Lisa painting.
> Exposed at the Louvre, the Mona Lisa is a portrait of Lisa Gherardini, wife of Francesco del Giocondo.

Describe the Mona Lisa painting.
> The Mona Lisa is a half-length portrait painting by the French Renaissance artist Leonardo da Vinci.
```

Which one is the best answer? Based on what criteria?

Supervised, non-LLM text generation metrics

We have a test set with **expected answers**, so we can use **supervised metrics**. This won't be the case in **production**!

String distance metrics are a good start (e.g. **Levenshtein**, **Hamming**...), but they don't really capture the **semantic similarity** between texts.

```
What is the capital of France?
> The capital of France is Paris. # expected answer
> Paris is the capital of France. # good answer, but different string

> Levenshtein distance = 20
```

Supervised, non-LLM text generation metrics

In the early 2000s, researches attempted to develop **more complex metrics** that would assess if a **translation** "captured the meaning" of a sentence compared to a set of reference translations.

- **BLEU**: Measures a **precision-type** score of the **overlap of n-grams** between the generated text and reference text (with a **brevity penalty**).
- **ROUGE**: Measures a **f1 or recall-type** score of the overlap of n-grams between the generated text and reference text.
- **METEOR**: Considers exact, stem, and synonym matches between the generated text and reference text, and computes a score based on these matches.

Using embeddings to compute similarity

Another approach is to use **embeddings** to compute distance between texts.

> An embedding is a **numerical representation** of a text in a high-dimensional space, where semantically similar texts are mapped to nearby points.

A quick history of text embeddings:

- **TF-IDF** and **Bag of Words**: Simple representations based on word frequency
- **Word Embeddings**: Dense vector representations of words (e.g., Word2Vec, GloVe)
- **LLMs**: Massive models that generate contextual embeddings which capture the meaning of words in context (e.g., BERT, GPT)

We can use **cosine similarity** or **euclidean distance** between embeddings to assess similarity between texts.

Supervised, LLM-as-a-judge metrics

A recent approach is to use LLMs themselves to evaluate the quality of generated text - these methods are called **LLM-as-a-judge**.

In a supervised context you can for example:

- Ask the LLM if the **facts** in the generated text are the same as the facts in the reference text.
- Ask the LLM if the text in the generated output is **coherent** and **relevant** to the input prompt.

A common practice on LLM-as-a-judge techniques is to make **several LLM calls** with slightly different prompts and average the results (**voting**).

Ragas

ragas is a Python package that implements the different metrics discussed.

```
from ragas.metrics import BleuScore

sample = SingleTurnSample(user_input="...",response="...", reference="...")

bleu_scorer = BleuScore()
bleu_scorer.single_turn_score(sample)
```

Access to an LLM can be passed through a LangChain LLM wrapper.

```
from langchain_google_genai import ChatGoogleGenerativeAI
from ragas.llm_wrappers import LangchainLLMWrapper

evaluator_llm = LangchainLLMWrapper(ChatGoogleGenerativeAI(model="gemini-2.5-flash"))
```

Any questions?

Onto the practice!