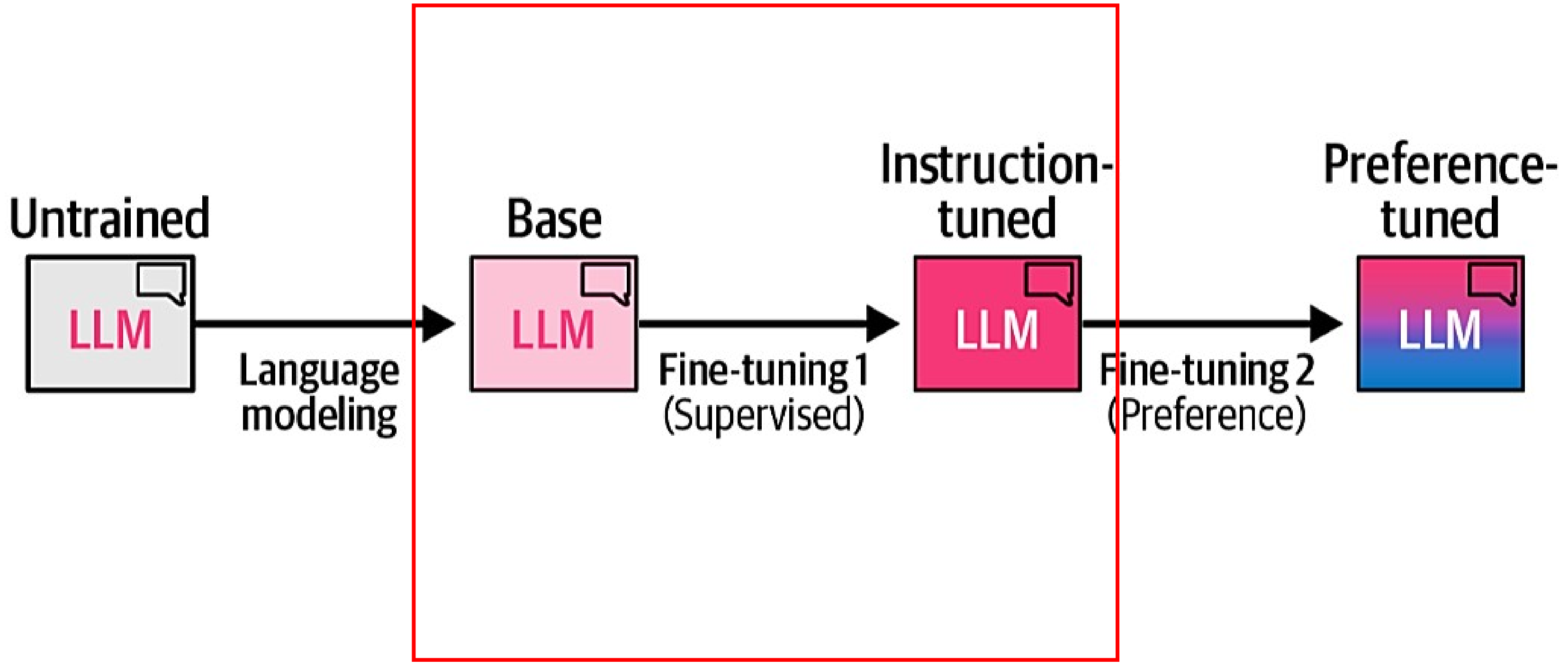
Parameter Efficient Finetuning (PEFT)

CS XXX: Introduction to Large Language Models

Contents

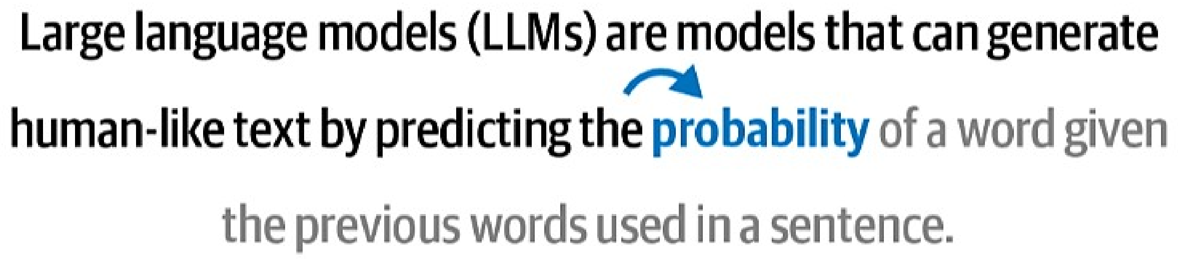
Three LLM Training Steps

- There are three common steps that lead to creating a high-quality LLM:



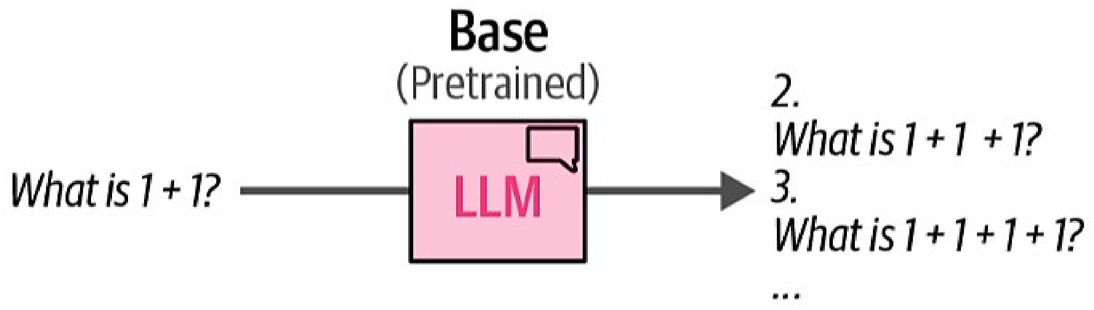
Three LLM Training Steps

- There are three common steps that lead to creating a high-quality LLM:
  - **Language modeling:**
    - The first step in creating a high-quality LLM is to pretrain it on one or more massive text datasets. During training, it attempts to predict the next token to accurately learn linguistic and semantic representations found in the text. As we have seen before, this is called language modelling and is a self-supervised method. During language modeling, the LLM aims to predict the next token based on an input. This is a process without labels.
    - This produces a base model, also commonly referred to as a pretrained or foundation model. Base models are a key artifact of the training process but are harder for the end user to deal with.


Large language models (LLMs) are models that can generate human-like text by predicting the **probability** of a word given the previous words used in a sentence.

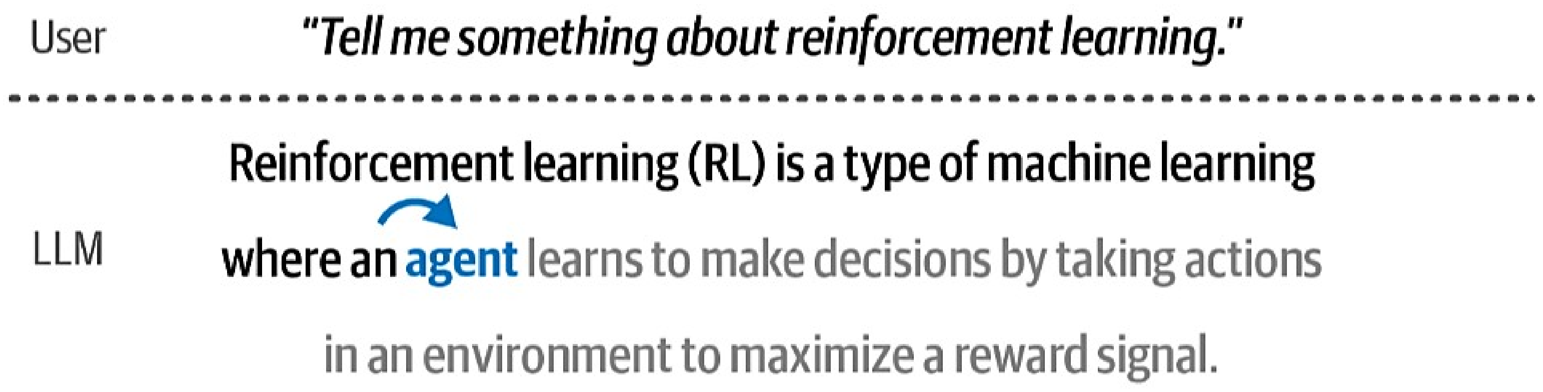
Three LLM Training Steps

- There are three common steps that lead to creating a high-quality LLM:
  - **Language modeling:**
    - A base LLM will not follow instructions but instead attempts to predict each next word. It may even create new questions.



Three LLM Training Steps

- There are three common steps that lead to creating a high-quality LLM:
  - **Fine-tuning 1 (supervised fine-tuning):**
    - LLMs are more useful if they respond well to instructions and try to follow them. With supervised fine-tuning (SFT), we can adapt the base model to follow instructions. During this fine-tuning process, the parameters of the base model are updated to be more in line with our target task, like following instructions. Like a pretrained model, it is trained using next-token prediction but instead of only predicting the next token, it does so based on a user input. the LLM aims to predict the next token based on an input that has additional labels. In a sense, the label is the user’s input.

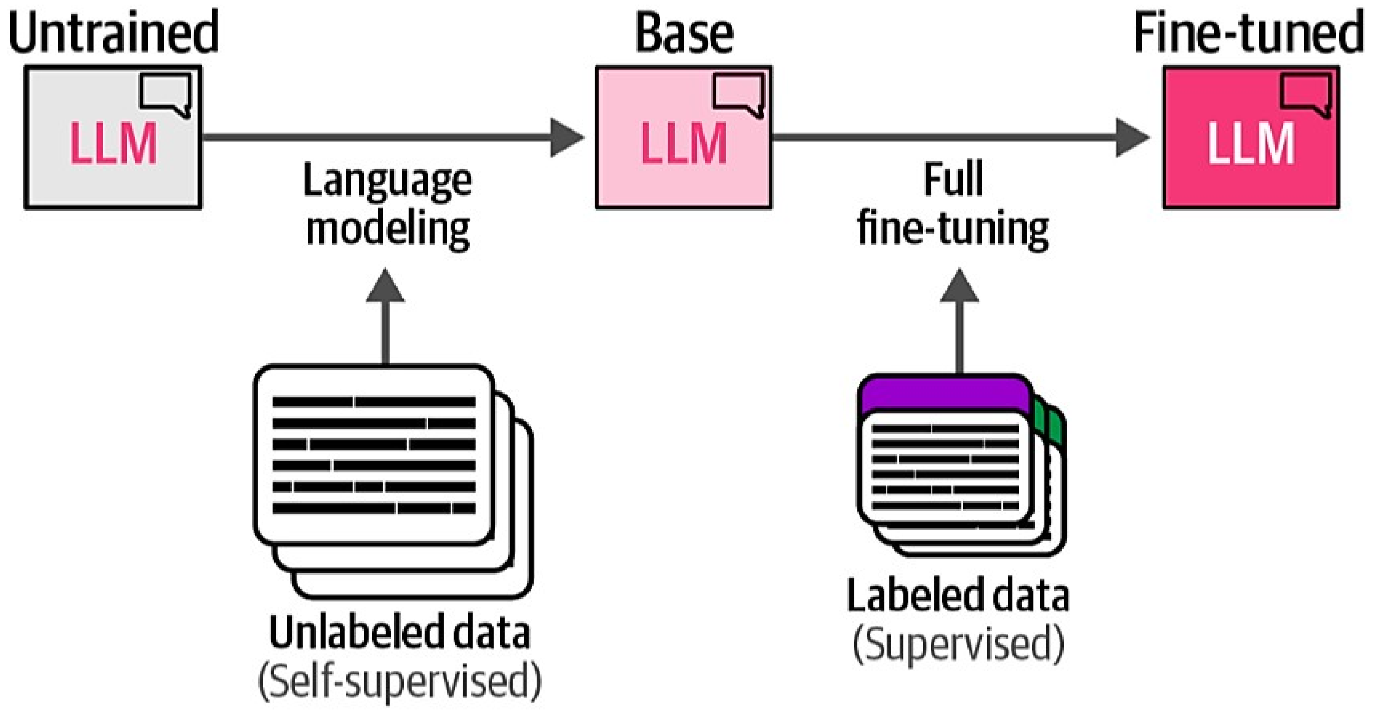

User
“Tell me something about reinforcement learning.”
LLM
Reinforcement learning (RL) is a type of machine learning
where an agent learns to make decisions by taking actions
in an environment to maximize a reward signal.

Three LLM Training Steps

- There are three common steps that lead to creating a high-quality LLM:
  - **Fine-tuning 2 (preference tuning):**
    - The final step further improves the quality of the model and makes it more aligned with the expected behavior of AI safety or human preferences. This is called preference tuning. Preference tuning is a form of fine-tuning and, as the name implies, aligns the output of the model to our preferences, which are defined by the data that we give it. Like SFT, it can improve upon the original model but has the added benefit of distilling preference of output in its training process.

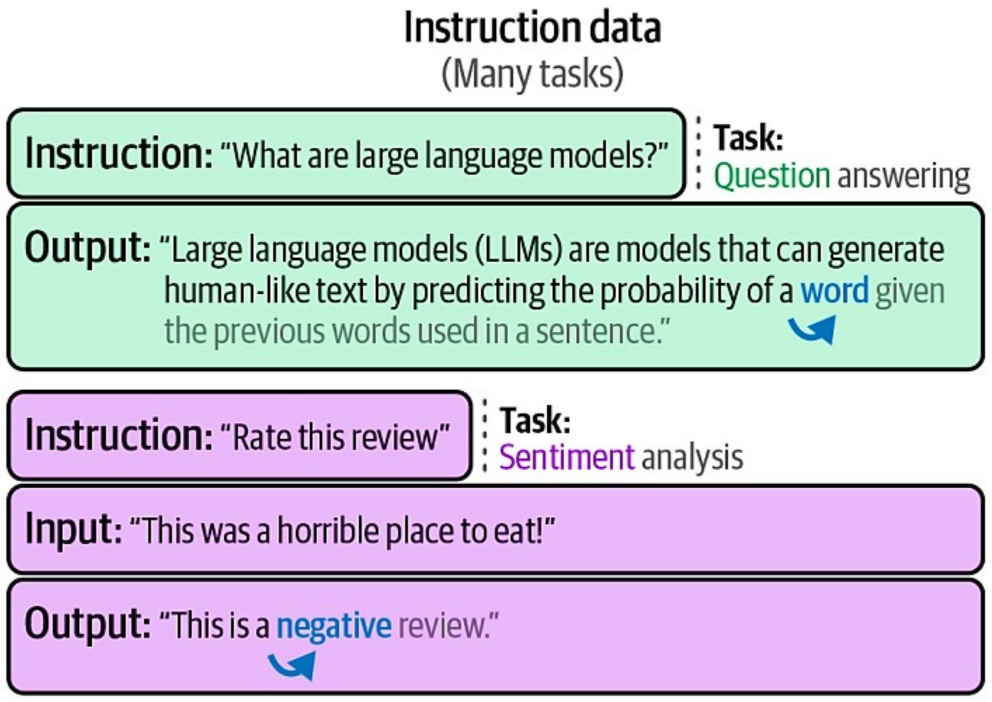
Full-Finetuning

- The most common fine-tuning process is full fine-tuning. Like pretraining an LLM, this process involves updating all parameters of a model to be in line with your target task. The main difference is that we now use a smaller but labeled dataset whereas the pretraining process was done on a large dataset without any labels



Full-Finetuning

- You can use any labeled data for full fine-tuning, making it also a great technique for learning domain-specific representations. To make our LLM follow instructions, we will need question-response data (Instruction data). This data is queries by the user with corresponding answers. The instructions can contain many different tasks.


Instruction data
(Many tasks)
Instruction: "What are large language models?"
Task: Question answering
Output: "Large language models (LLMs) are models that can generate human-like text by predicting the probability of a word given the previous words used in a sentence."
Instruction: "Rate this review"
Task: Sentiment analysis
Input: "This was a horrible place to eat!"
Output: "This is a negative review."

Parameter Efficient Finetuning (PEFT)

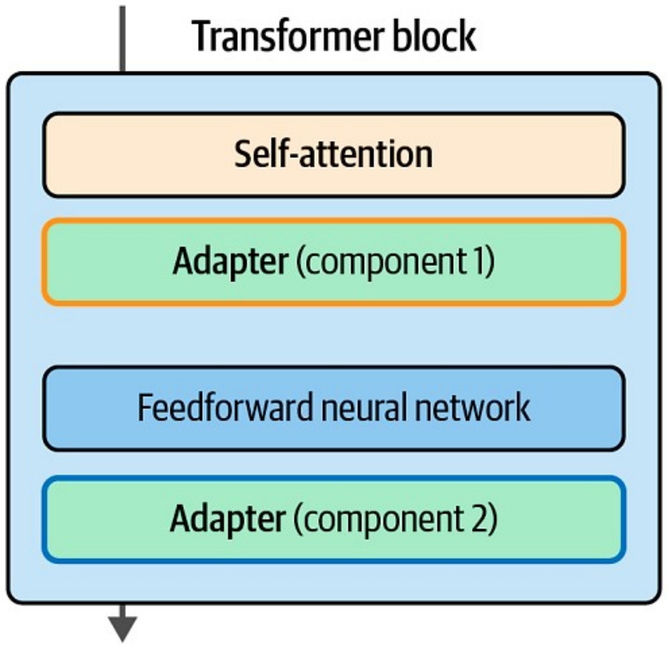
- Updating all parameters of a model has a large potential of increasing its performance but comes with several disadvantages. It is costly to train, has slow training times, and requires significant storage.
- To resolve these issues, attention has been given to parameter-efficient fine-tuning (PEFT) alternatives that focus on fine-tuning pre-trained models at higher computational efficiency.

PEFT – Adapters

- Adapters are a core component of many PEFT-based techniques. The method proposes a set of additional modular components inside the Transformer that can be fine-tuned to improve the model’s performance on a specific task without having to fine-tune all the model weights. This saves a lot of time and compute.
- Adapters are described in the paper “Parameter-efficient transfer learning for NLP”, which showed that fine-tuning 3.6% of the parameters of BERT for a task can yield comparable performance to fine-tuning all the model’s weights.

PEFT – Adapters

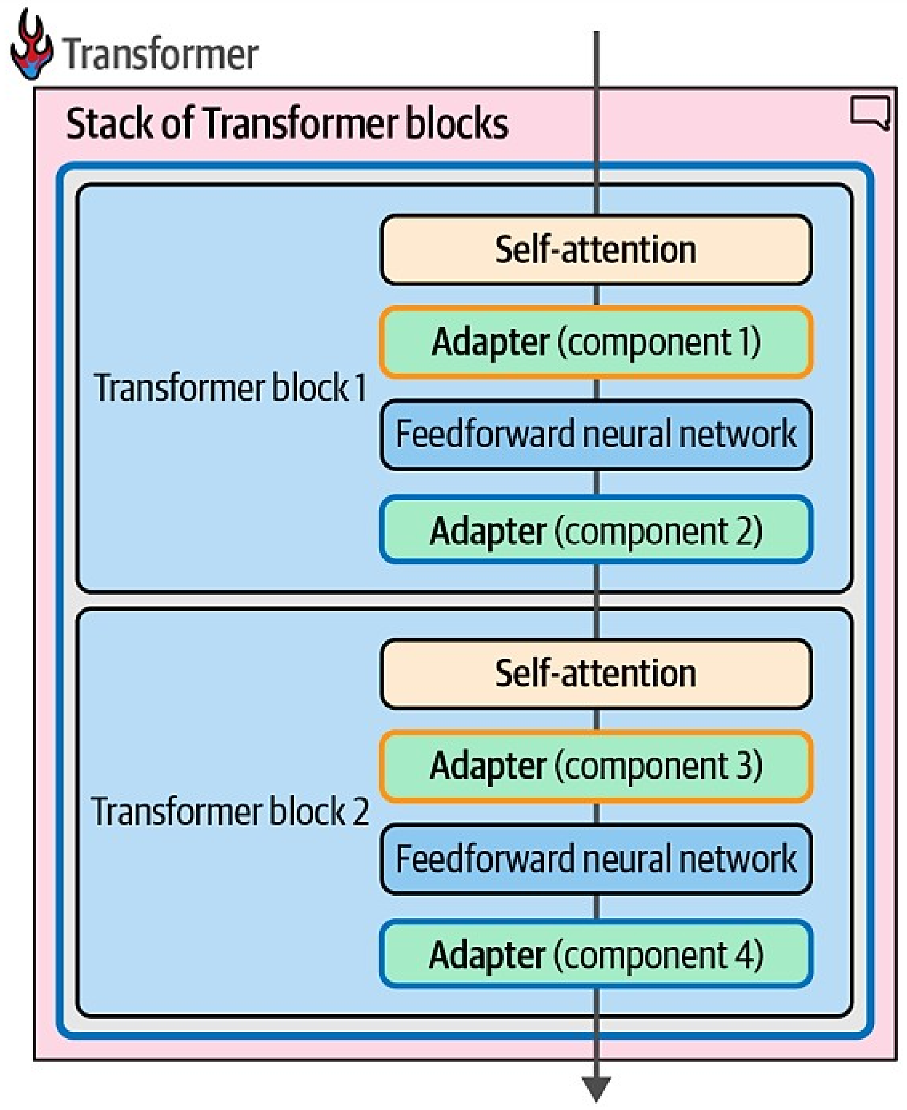
- In a single Transformer block, the paper’s proposed architecture places adapters after the attention layer and the feedforward neural network.



- Adapters add a small number of weights in certain places in the network that can be fine-tuned efficiently while leaving the majority of model weights frozen.

PEFT – Adapters

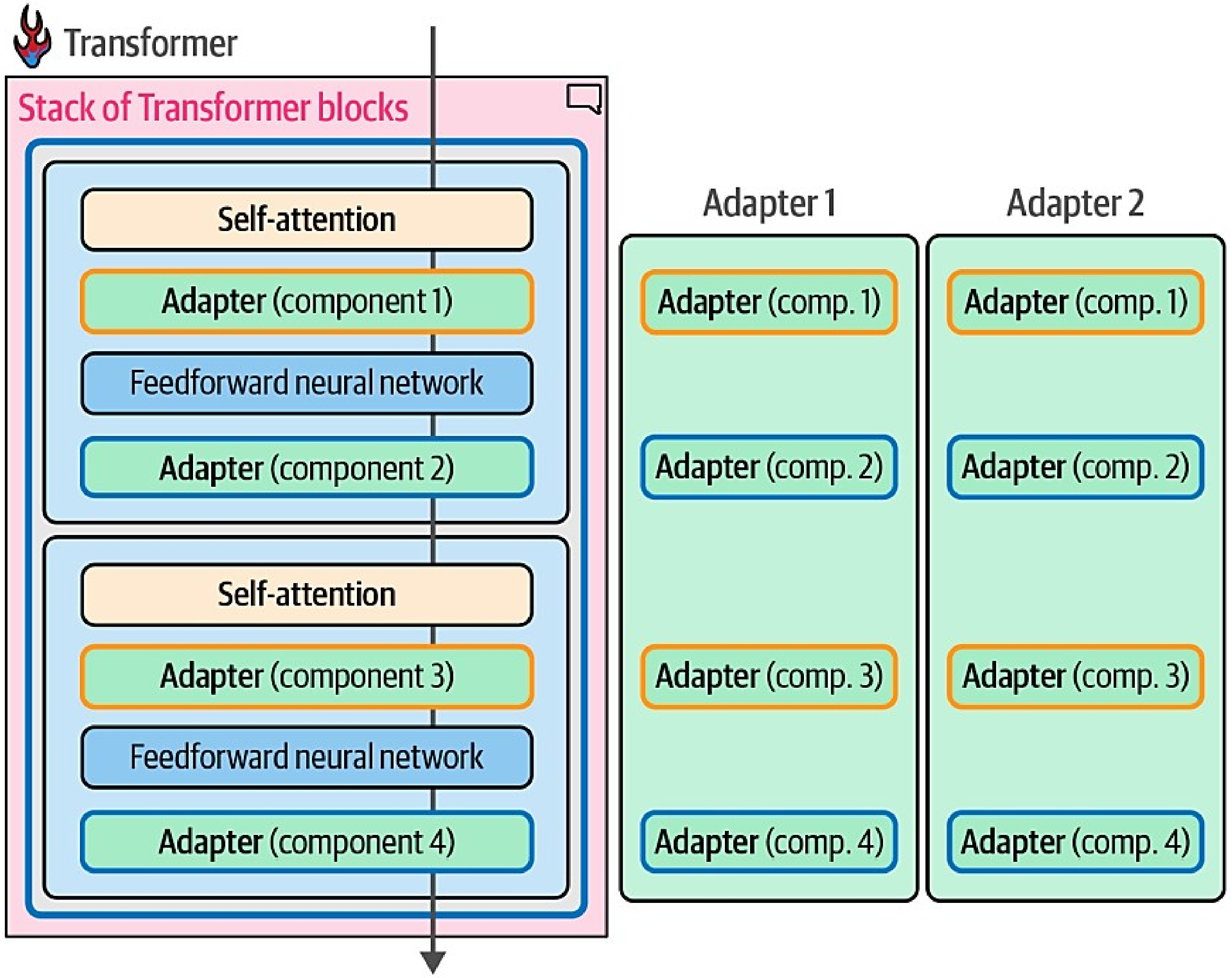
- Adapter components span the various Transformer blocks in the model.



Adapter 1 can be a specialist in, say, medical text classification, while **Adapter 2** can specialize in named-entity recognition (NER)

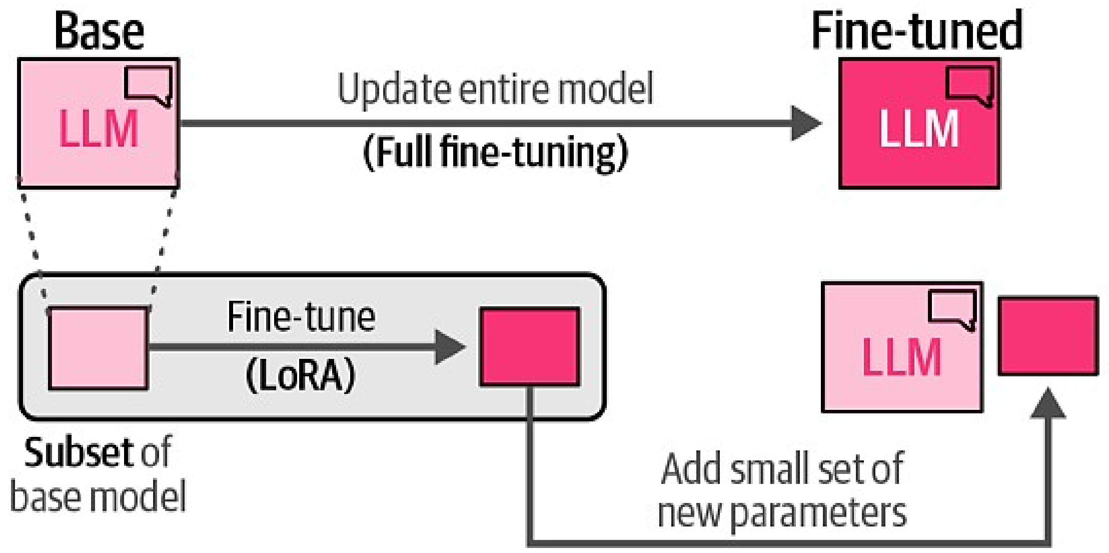
PEFT – Adapters

- Adapters that specialize in specific tasks can be swapped into the same architecture



PEFT – Low-Rank Adaptation (LoRA)

- As an alternative to adapters, low-rank adaptation (LoRA) was introduced and is at the time of writing is a widely used and effective technique for PEFT. LoRA is a technique that (like adapters) only requires updating a small set of parameters.
- LoRA creates a small subset of the base model to fine-tune instead of adding layers to the model

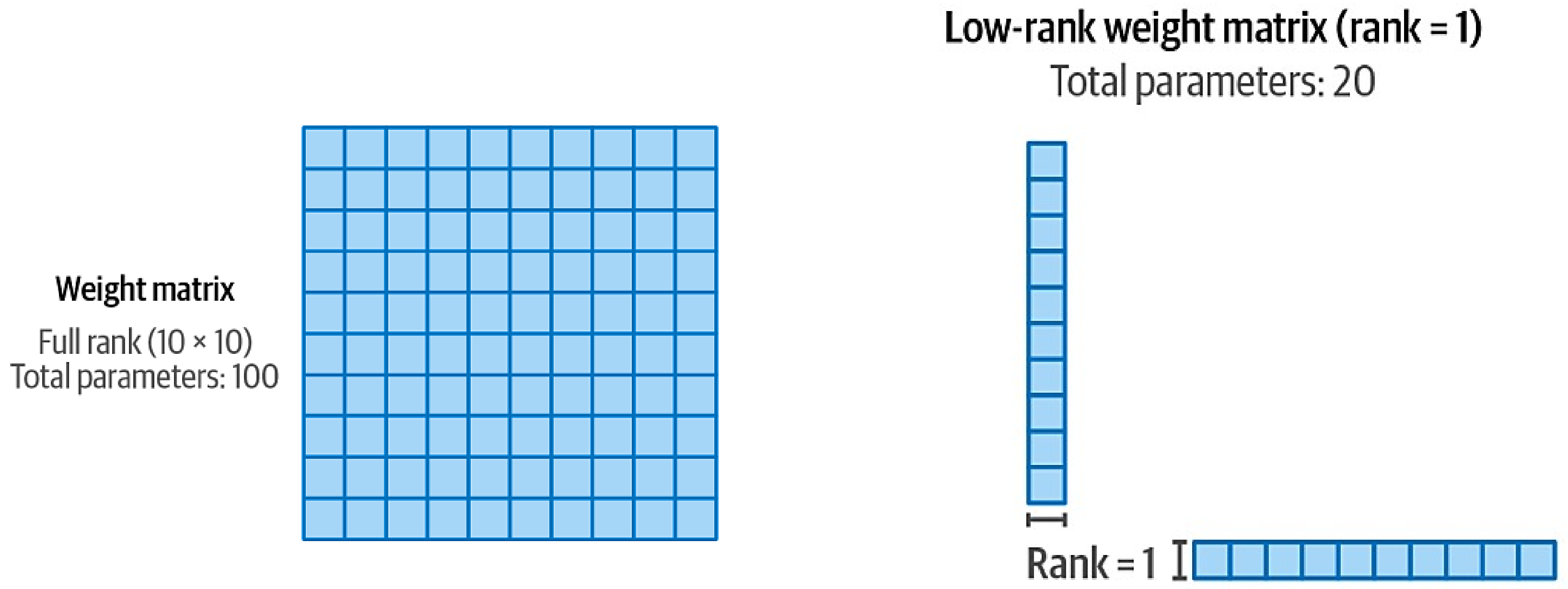


PEFT – Low-Rank Adaptation (LoRA)

- Like adapters, this subset allows for much quicker fine-tuning since we only need to update a small part of the base model.
- We create this subset of parameters by approximating large matrices that accompany the original LLM with smaller matrices. We can then use those smaller matrices as a replacement and fine-tune them instead of the original large matrices.

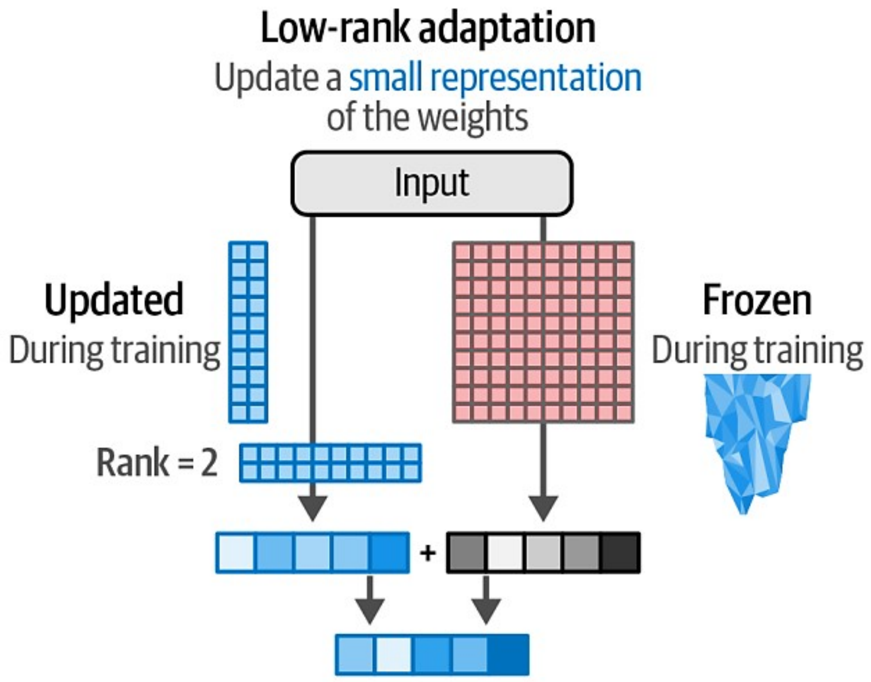
PEFT – Low-Rank Adaptation (LoRA)

- Take for example the 10 × 10 matrix
- We can come up with two smaller matrices, which when multiplied, reconstruct a 10 × 10 matrix. This is a major efficiency win because instead of using 100 weights (10 times 10) we now only have 20 weights (10 plus 10)



PEFT – Low-Rank Adaptation (LoRA)

- Decomposing a large weight matrix into two smaller matrices leads to a compressed, low-rank version of the matrix that can be fine-tuned more efficiently
- During training, we only need to update these smaller matrices instead of the full weight changes. The updated change matrices (smaller matrices) are then combined with the full (frozen) weights



PEFT – Low-Rank Adaptation (LoRA)

- Papers like "Intrinsic dimensionality explains the effectiveness of language model fine-tuning" demonstrate that language models "have a very low intrinsic dimension." This means that we can find small ranks that approximate even the massive matrices of an LLM. A 175B model like GPT-3, for example, would have a weight matrix of 12,288 × 12,288 inside each of its 96 Transformer blocks. That's 150 million parameters. If we can successfully adapt that matrix into rank 8, that would only require two 12,288 × 2 matrices resulting in 197K parameters per block. These are major savings in speed, storage.

- This smaller representation is quite flexible in that you can select which parts of the base model to fine-tune. For instance, we can only fine-tune the Query and Value weight matrices in each Transformer layer.

PEFT – Low-Rank Adaptation (LoRA)

- Papers like “Intrinsic dimensionality explains the effectiveness of language model fine-tuning” demonstrate that language models “have a very low intrinsic dimension.” This means that we can find small ranks that approximate even the massive matrices of an LLM. A 175B model like GPT-3, for example, would have a weight matrix of 12,288 × 12,288 inside each of its 96 Transformer blocks. That’s 150 million parameters. If we can successfully adapt that matrix into rank 8, that would only require two 12,288 × 2 matrices resulting in 197K parameters per block. These are major savings in speed, storage.

- This smaller representation is quite flexible in that you can select which parts of the base model to fine-tune. For instance, we can only fine-tune the Query and Value weight matrices in each Transformer layer.

QLoRA

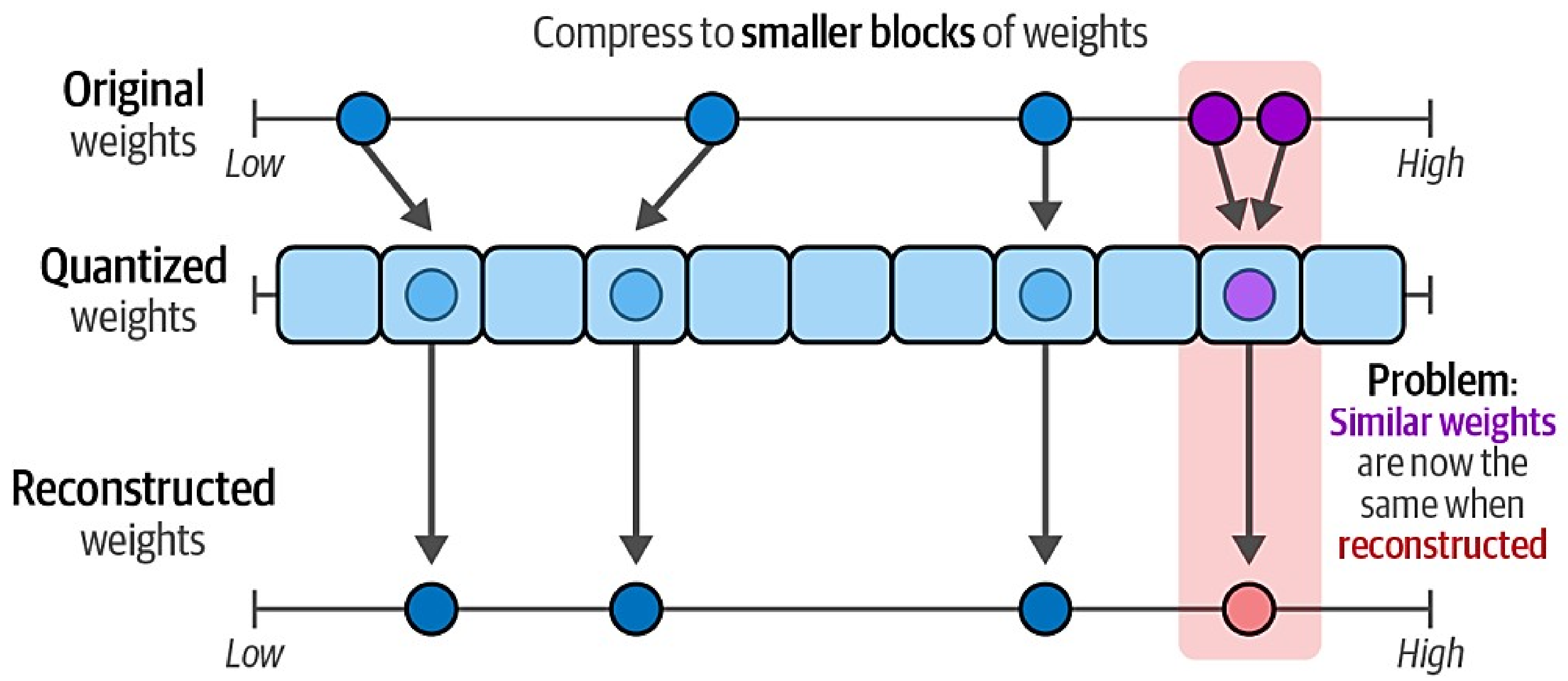
- We can make LoRA even more efficient by reducing the memory requirements of the model’s original weights before projecting them into smaller matrices.

RECALL

- With quantization, we aim to lower the number of bits while still accurately representing the original weight values

QLoRA

- When directly mapping higher precision values to lower precision values, multiple higher precision values might end up being represented by the same lower precision values.


Compress to smaller blocks of weights
Original weights
Low
High
Quantized weights
Problem: Similar weights are now the same when reconstructed
Reconstructed weights
Low
High

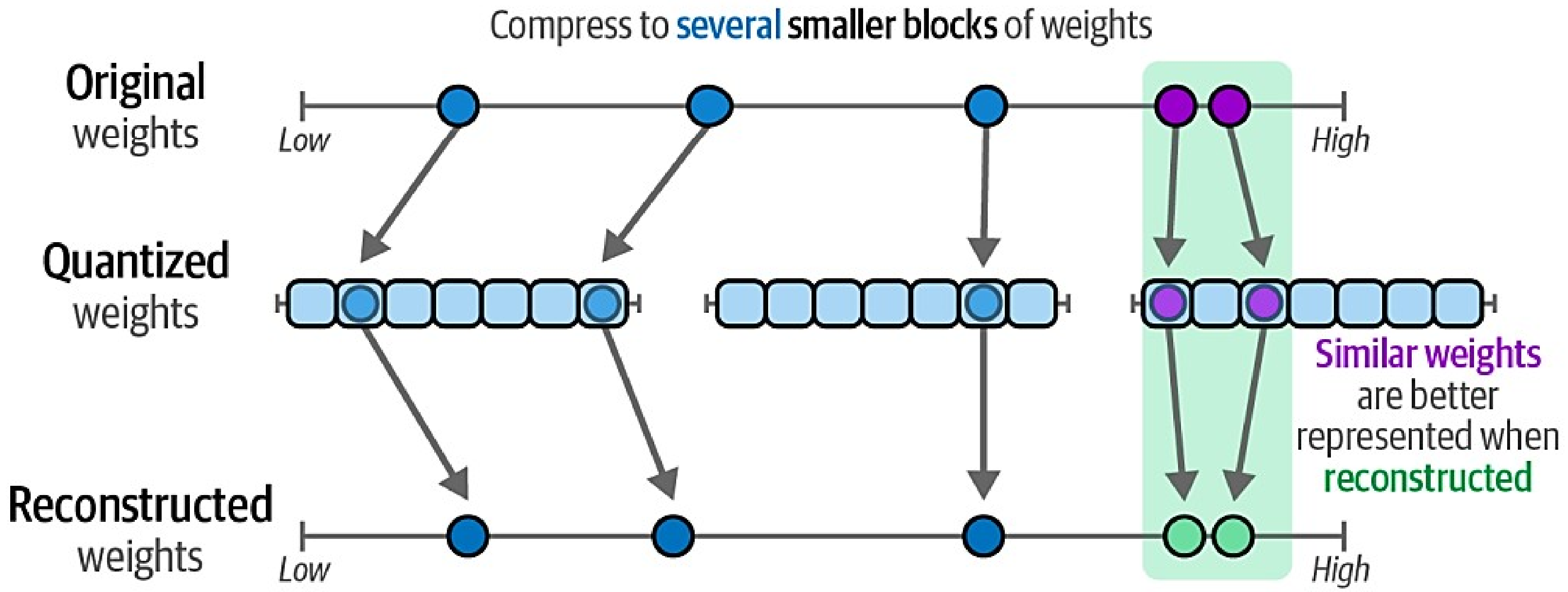
Quantizing weights that are close to one another results in the same reconstructed weights thereby removing any differentiating factor.

QLoRA

- Instead, the authors of QLoRA, a quantized version of LoRA, found a way to go from a higher number of bits to a lower value and vice versa without differentiating too much from the original weights.
- They used blockwise quantization to map certain blocks of higher precision values to lower precision values. Instead of directly mapping higher precision to lower precision values, additional blocks are created that allow for quantizing similar weights.

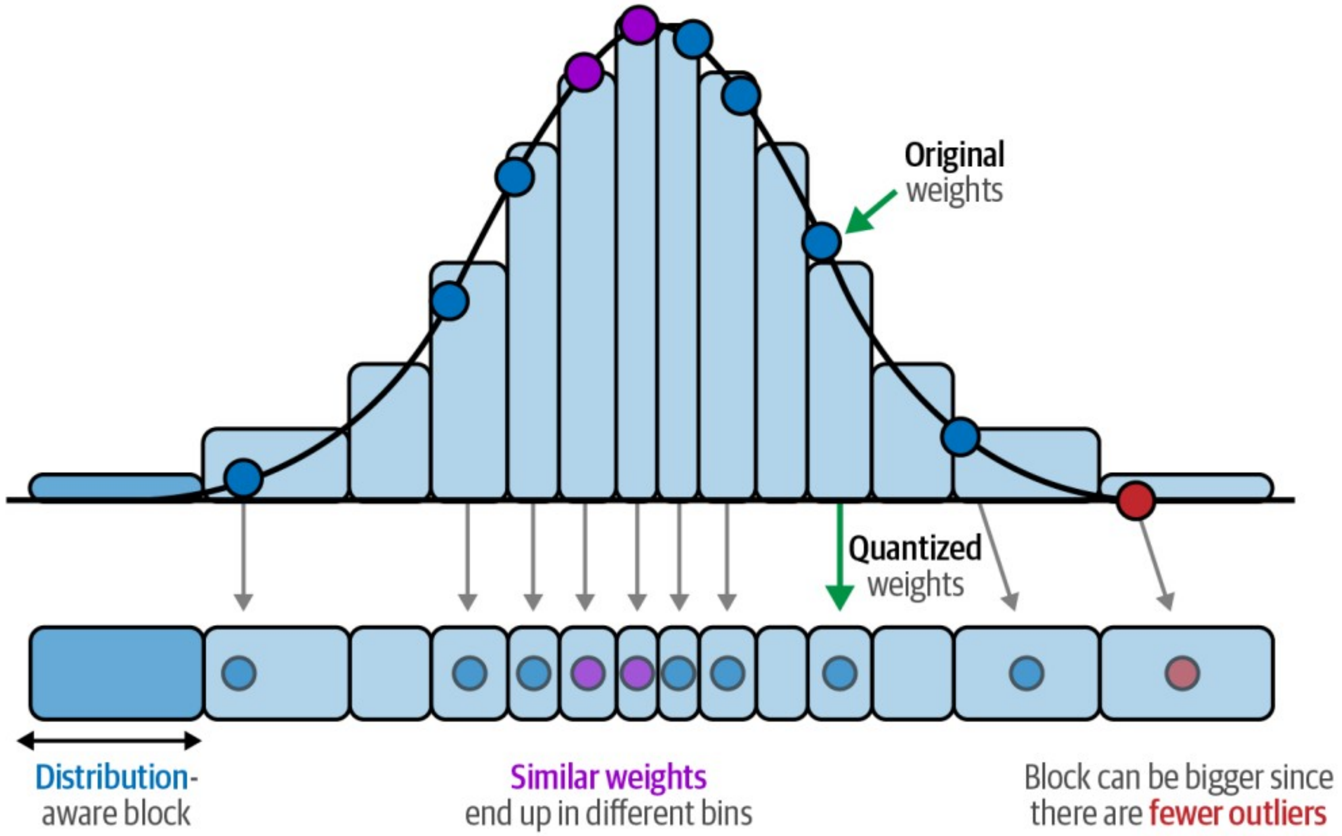
QLoRA

- Blockwise quantization can accurately represent weights in lower precision through quantization blocks


Compress to several smaller blocks of weights
Original weights
Low
High
Quantized weights
Similar weights are better represented when reconstructed
Reconstructed weights
Low
High

QLoRA

- A nice property of neural networks is that their values are generally normally distributed between –1 and 1. This property allows us to bin the original weights to lower bits based on their relative density.
- Using distribution-aware blocks we can prevent values close to one another from being represented with the same quantized value.


Original weights
Quantized weights
Distribution-aware block
Similar weights end up in different bins
Block can be bigger since there are fewer outliers

QLoRA

- Combined with the blockwise quantization, this normalization procedure allows for accurate representation of high precision values by low precision values with only a small decrease in the performance of the LLM. As a result, we can go from a 16-bit float representation to a measly 4-bit normalized float representation. A 4-bit representation significantly reduces the memory requirements of the LLM during training.

QLoRA

- Combined with the blockwise quantization, this normalization procedure allows for accurate representation of high precision values by low precision values with only a small decrease in the performance of the LLM. As a result, we can go from a 16-bit float representation to a measly 4-bit normalized float representation. A 4-bit representation significantly reduces the memory requirements of the LLM during training.

References

- Alammar, J., & Grootendorst, M. Hands-On Large Language Models: Language Understanding and Generation. O'Reilly Media.