Large Language Model Lora Training

Yen-Ting Lin 林彥廷

Nov 16, 2023 @ ADL Recitation, NTU

LLM Development

LLM Development

Homework 3

Instruction tuning

Can you run it?

Access token

Model name (Press Enter to apply)

mistralai/Mistral-7B-v0.1

GPU Vendor

NVIDIA

Filter by RAM (GB)

10.00 40.00

0.50 96.00

GPU

A10 PCIe

LoRa % trainable parameters

2.00

0.10 100.00

	INFO
Product Name	A10 PCIe
GPU Chip	GA102
Released	Apr 12th, 2021
Bus	PCIe 4.0 x16
Memory	24 GB, GDDR6, 384 bit

Can you run it? LLM version

Information

[mistralai/Mistral-7B-v0.1](https://huggingface.co/mistralai/Mistral-7B-v0.1) (7.2B)

int4 int8 float16/bfloat16 float32

`int4` refers to models in `GPTQ-4bit`, `AWQ-4bit` or `Q4_0 GGUF/GGML`

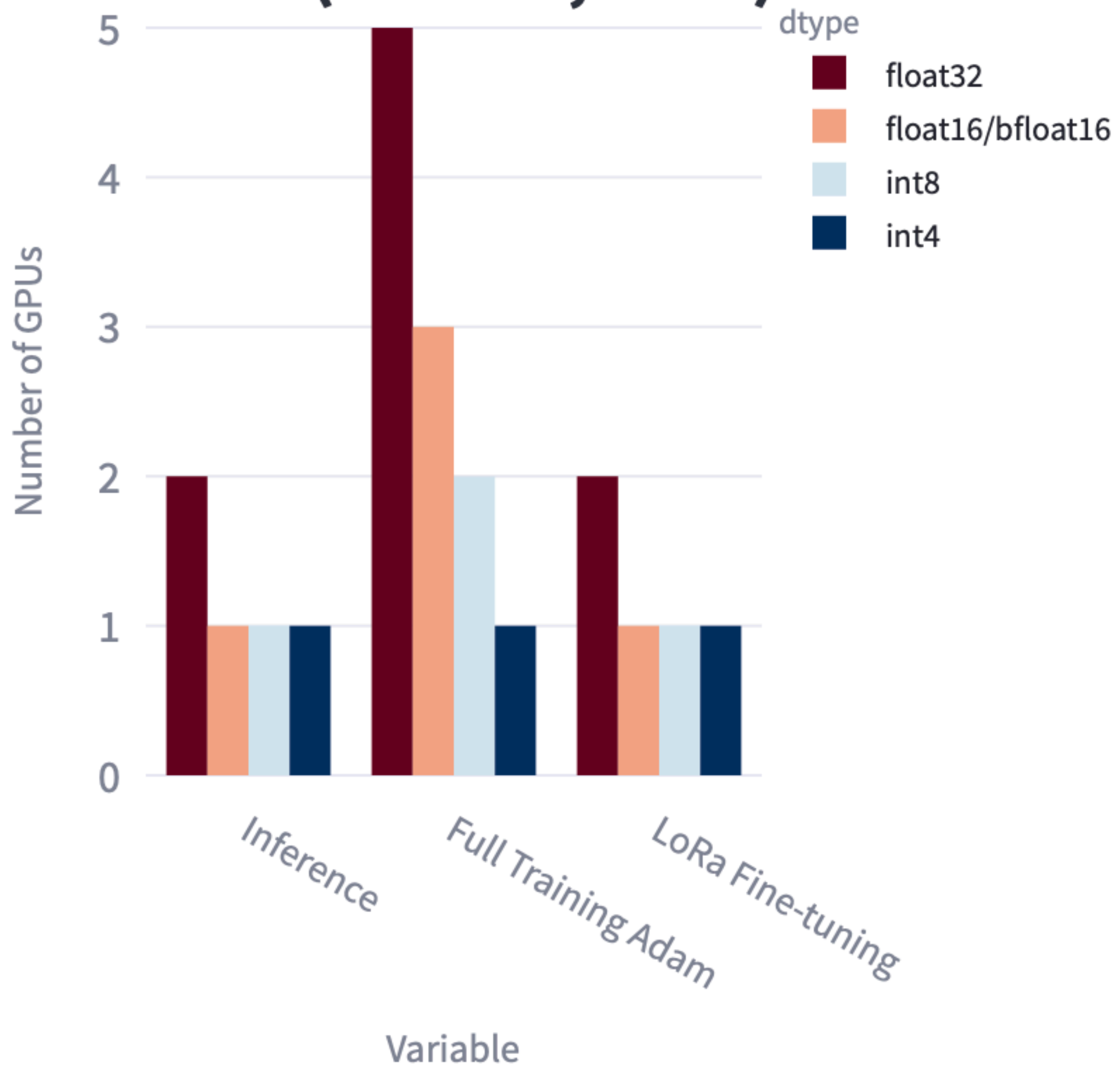
✅ You require **1** GPUs for **Inference**

✅ You require **1** GPUs for **Full Training Adam**

✅ You require **1** GPUs for **LoRa Fine-tuning** (2.0%)

	float32	float16/bflo
Total Size (GB)	27.49	1
Inference (GB)	32.99	1
Training using Adam (GB)	109.96	5

Number of GPUs required for A10 PCIe (24.0 GB, 2021)



Source: https://huggingface.co/spaces/Vokturz/can-it-run-llm

Motivation

Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning

Armen Aghajanyan
Facebook AI
armenag@fb.com

Sonal Gupta
Facebook
sonalgupta@fb.com

Luke Zettlemoyer
Facebook AI
University of Washington
lsz@fb.com

- Core Finding: **Low Intrinsic Dimensionality in Language Models**
- Significance of Intrinsic Dimensionality:
  - Intrinsic dimensionality is a crucial metric that explains why large language models are efficiently fine-tunable with limited data.
- Broader Impact:
  - Understanding intrinsic dimensionality could lead to more resource-efficient and effective ways to train and deploy language models.

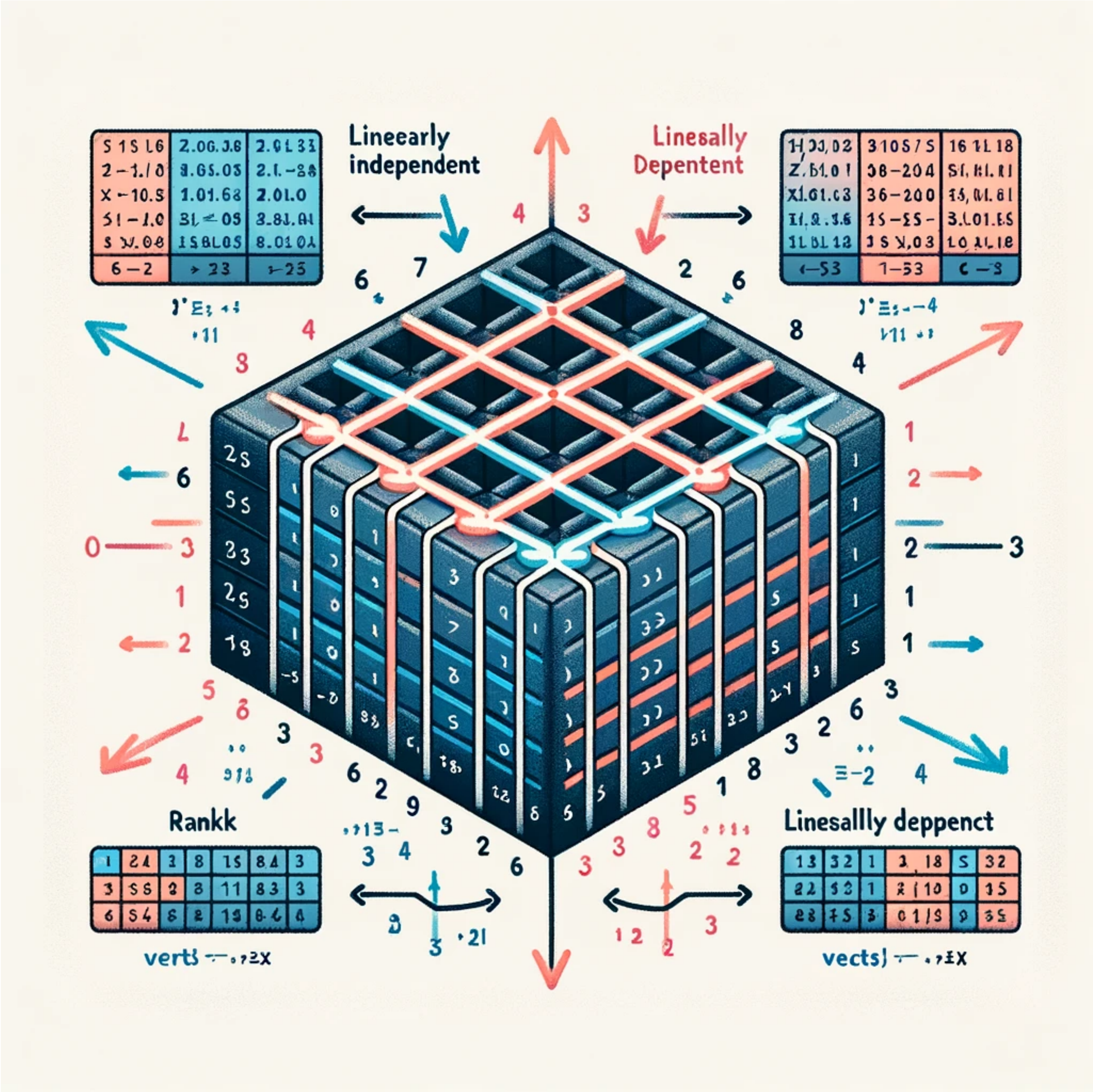
Rank

$$\begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} \xrightarrow{2R_1+R_2\to R_2} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 3 & 5 & 0 \end{bmatrix} \xrightarrow{-3R_1+R_3\to R_3} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & -1 & -3 \end{bmatrix}$$

$$\xrightarrow{R_2+R_3\to R_3} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{-2R_2+R_1\to R_1} \begin{bmatrix} 1 & 0 & -5 \\ 0 & 1 & 3 \\ 0 & 0 & 0 \end{bmatrix}.$$

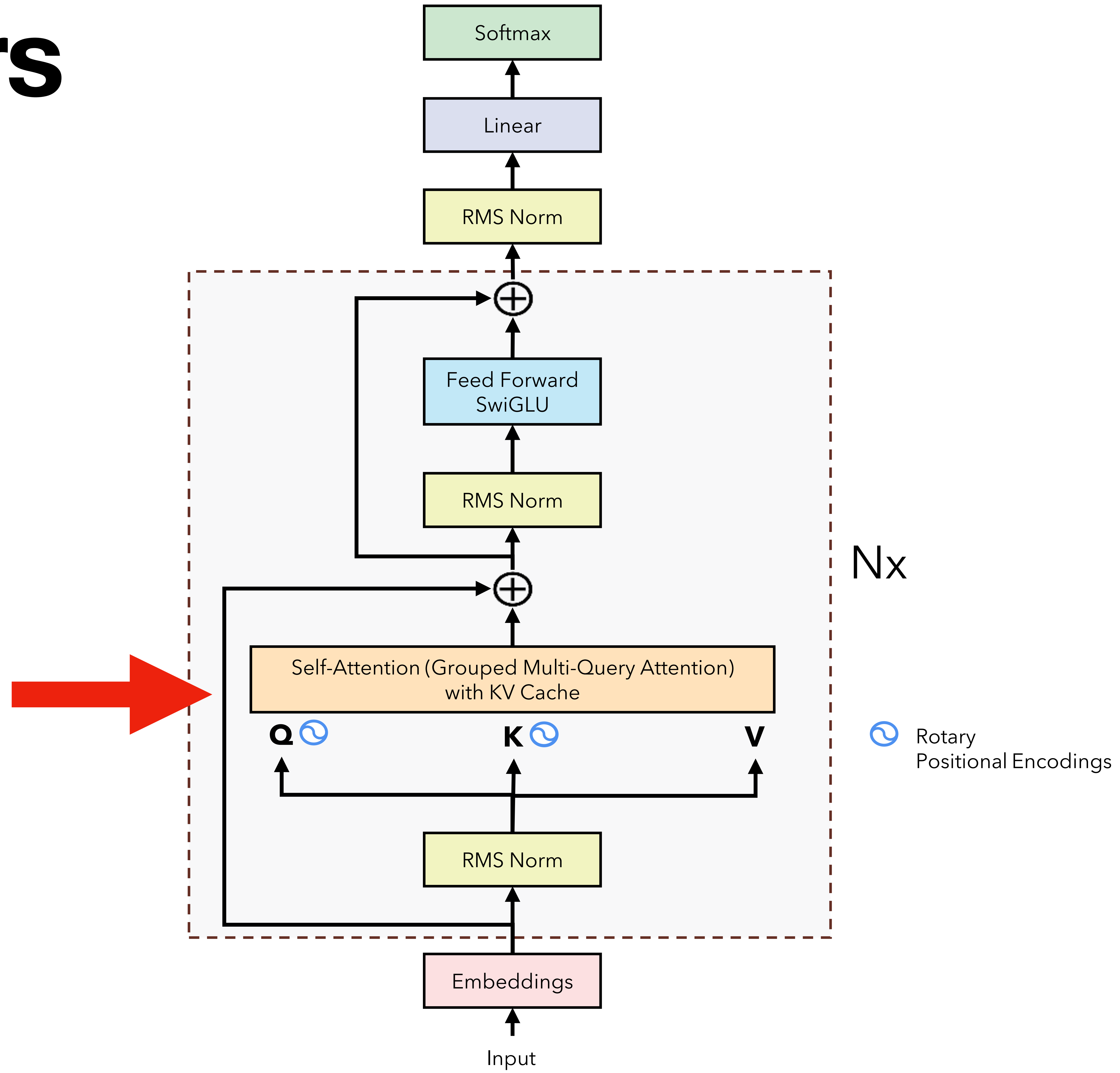
Source: https://en.wikipedia.org/wiki/Rank_(linear_algebra)

Rank



Lineearly independent
Linesally Depentent
Rankk
Linesallly deppenct

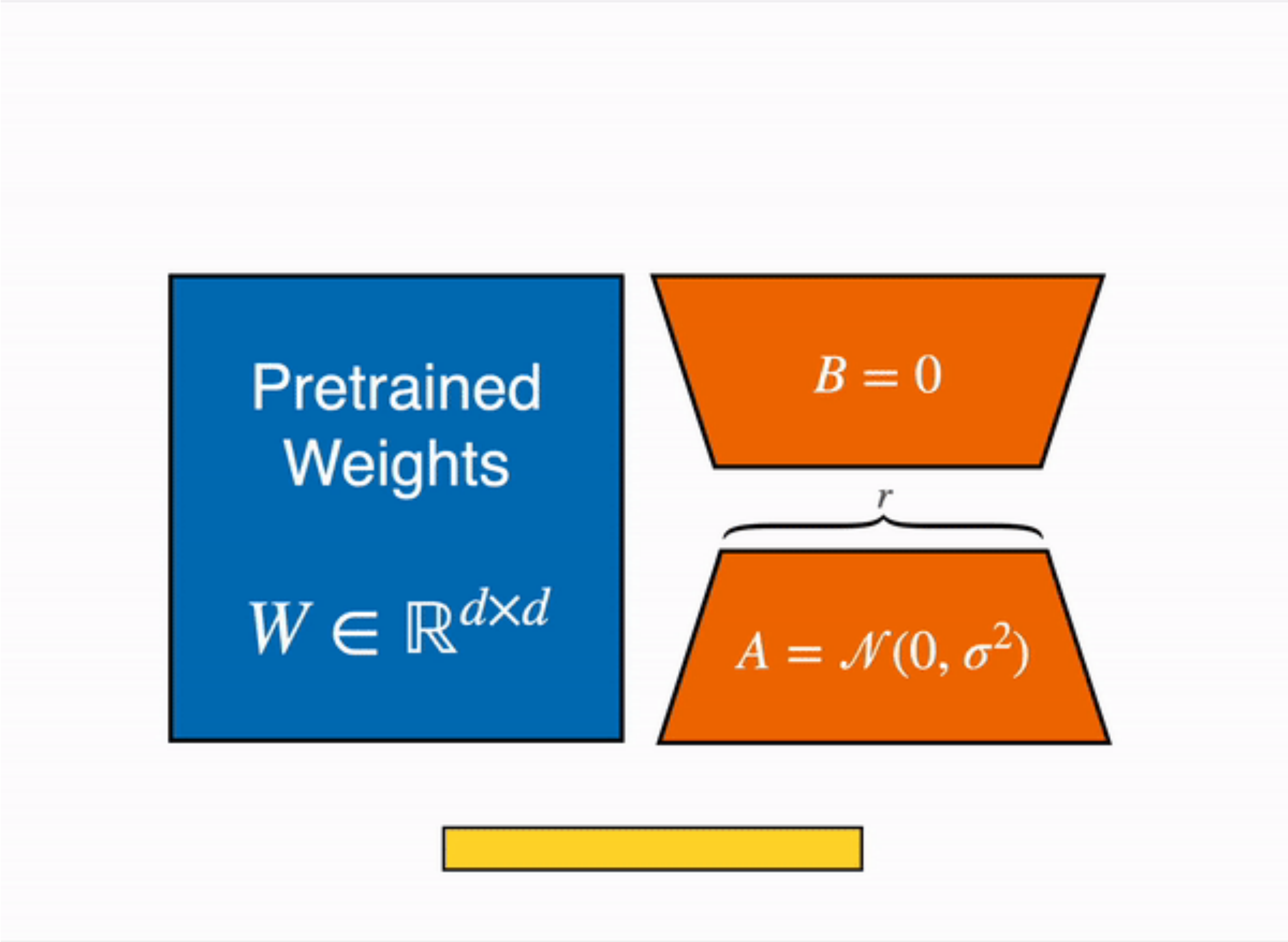
Transformers



LLaMA

Umar Jamil – https://github.com/hkproj/pytorch-llama-notes

Lora



Source: https://huggingface.co/blog/4bit-transformers-bitsandbytes

Lora

- Original Parameter: **W (d*d)**
- Introduce two new metrics **A (d, r)** and **B (r, d)**
- **r** is usually between 1 to 32

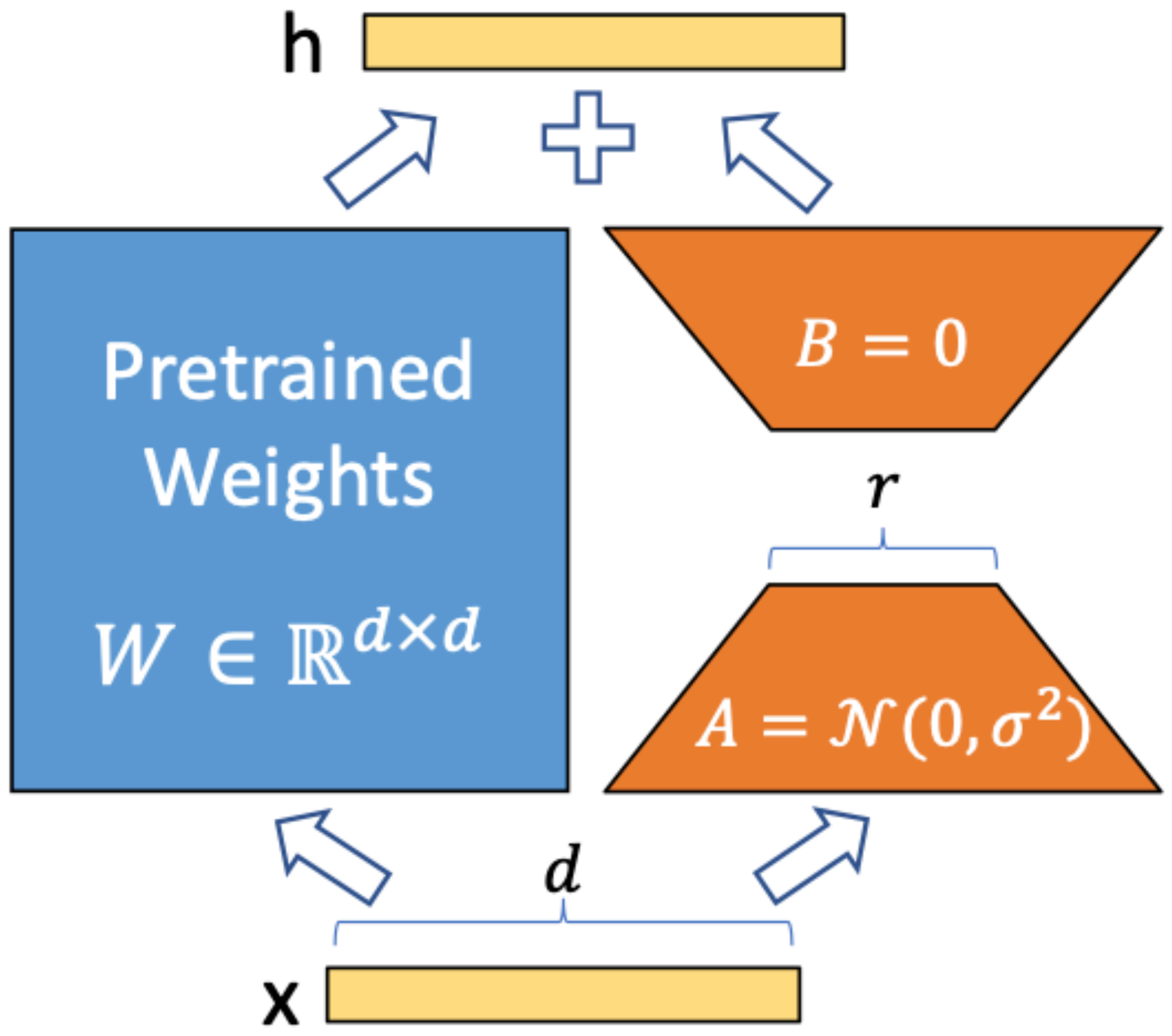


Figure 1: Our reparametrization. We only train A and B.

Source: https://arxiv.org/pdf/2106.09685.pdf

Benefits of Lora

- Less memory
  - no gradients for pretrained weights
- Plug-and-Play Lora

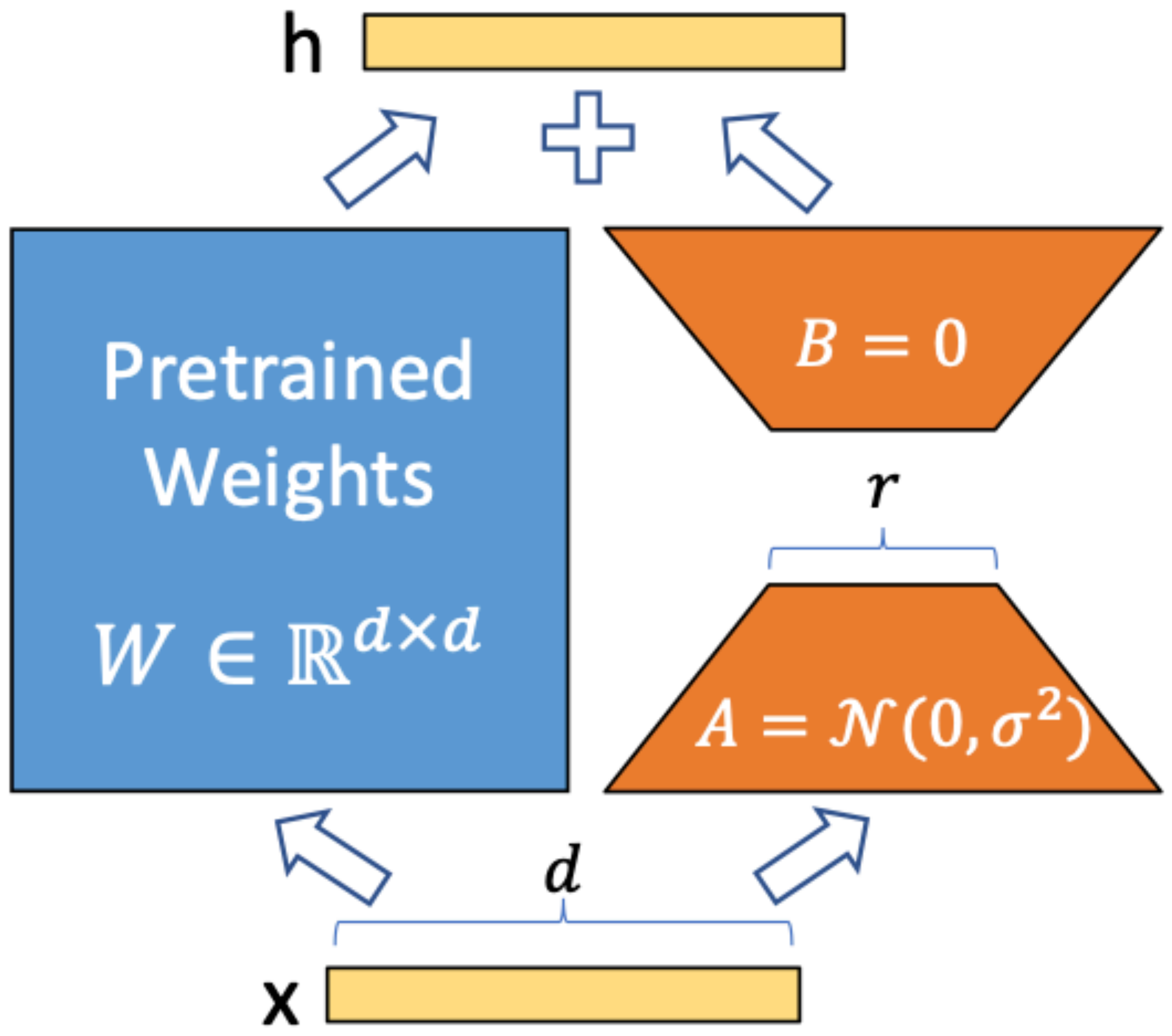


Figure 1: Our reparametrization. We only train A and B.

Source: https://arxiv.org/pdf/2106.09685.pdf

Which Weight Metrics?

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both W_q and W_v gives the best performance overall. We find the standard deviation across random seeds to be consistent for a given dataset, which we report in the first column.

Source: https://arxiv.org/pdf/2106.09685.pdf

Optimal Rank?

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Source: https://arxiv.org/pdf/2106.09685.pdf

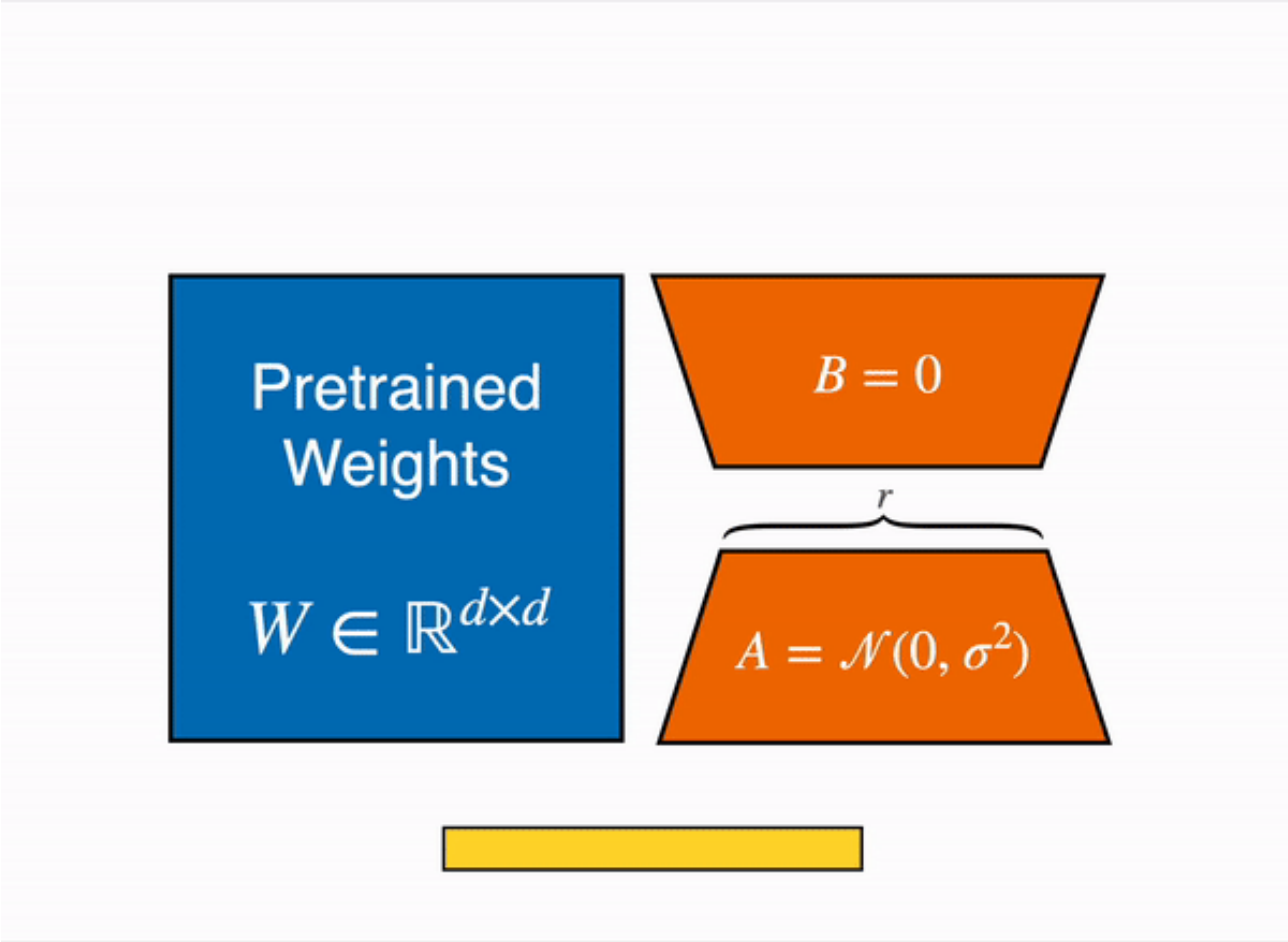
PEFT Comparion on GPT-3

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{H})	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm0.5\%$, MNLI-m around $\pm0.1\%$, and SAMSum around $\pm0.2/\pm0.2/\pm0.1$ for the three metrics.

Source: https://arxiv.org/pdf/2106.09685.pdf

Lora



Source: https://huggingface.co/blog/4bit-transformers-bitsandbytes

Constraint of Lora

- Pretrained Weights still account for large memory space

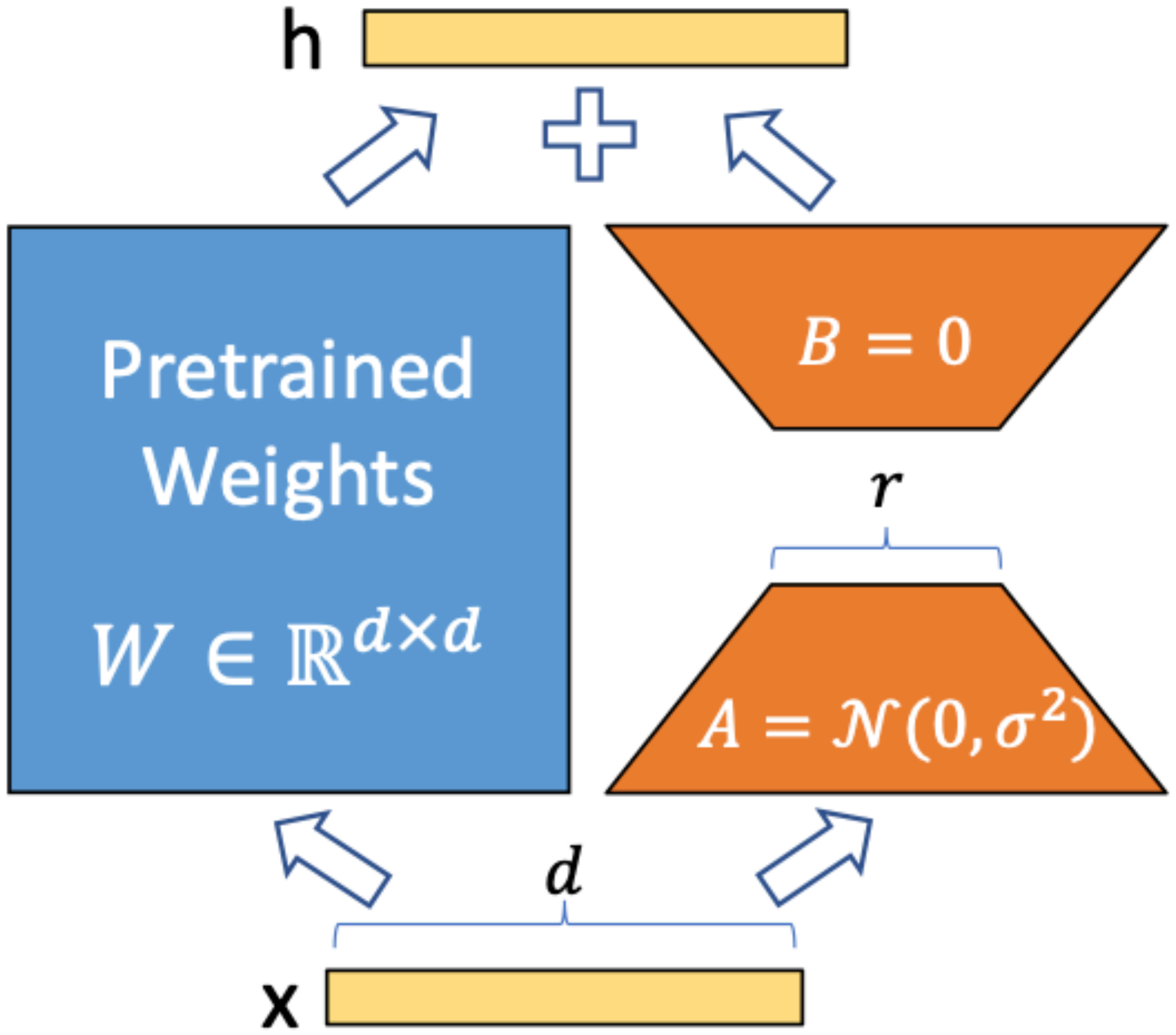
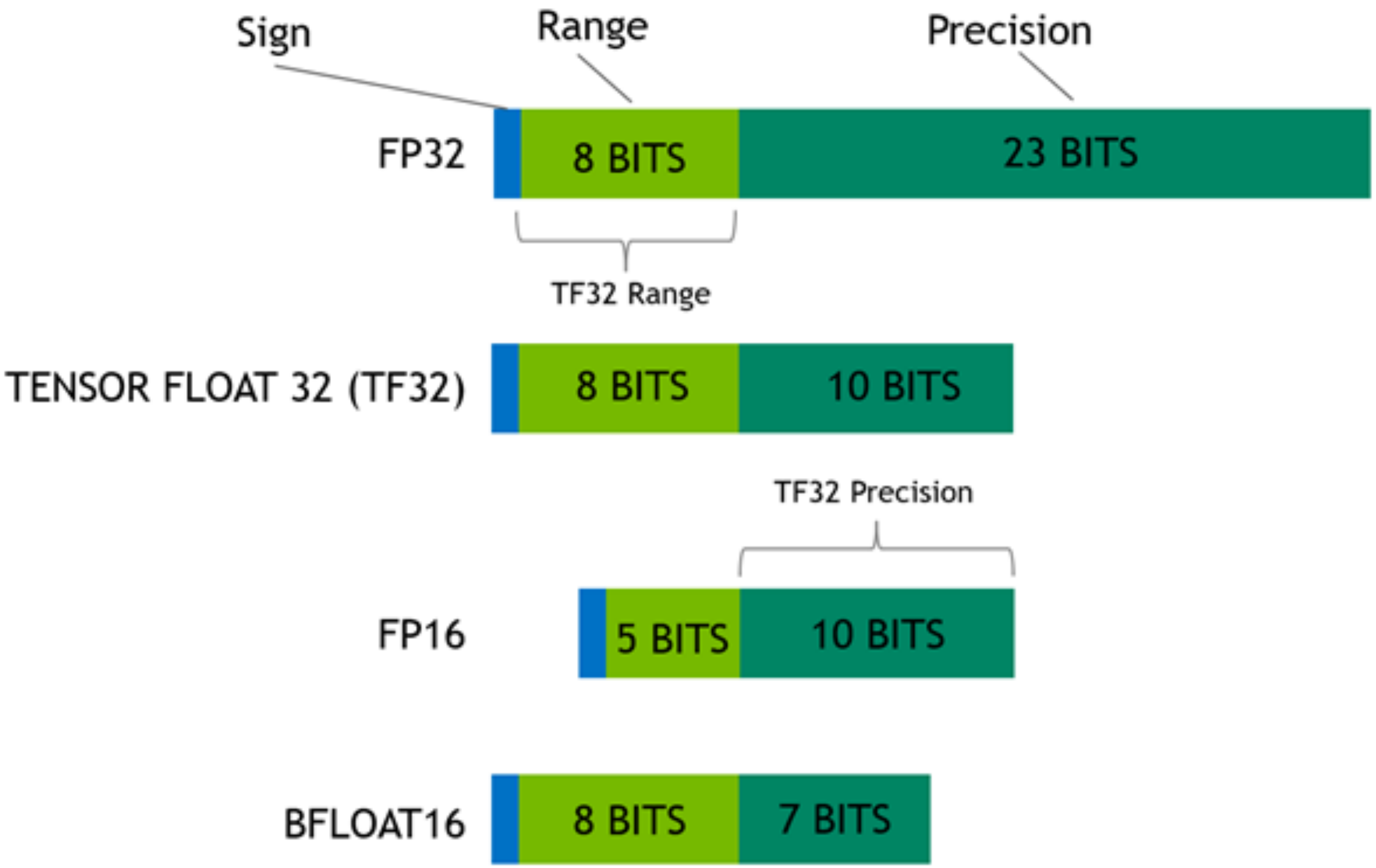


Figure 1: Our reparametrization. We only train A and B.

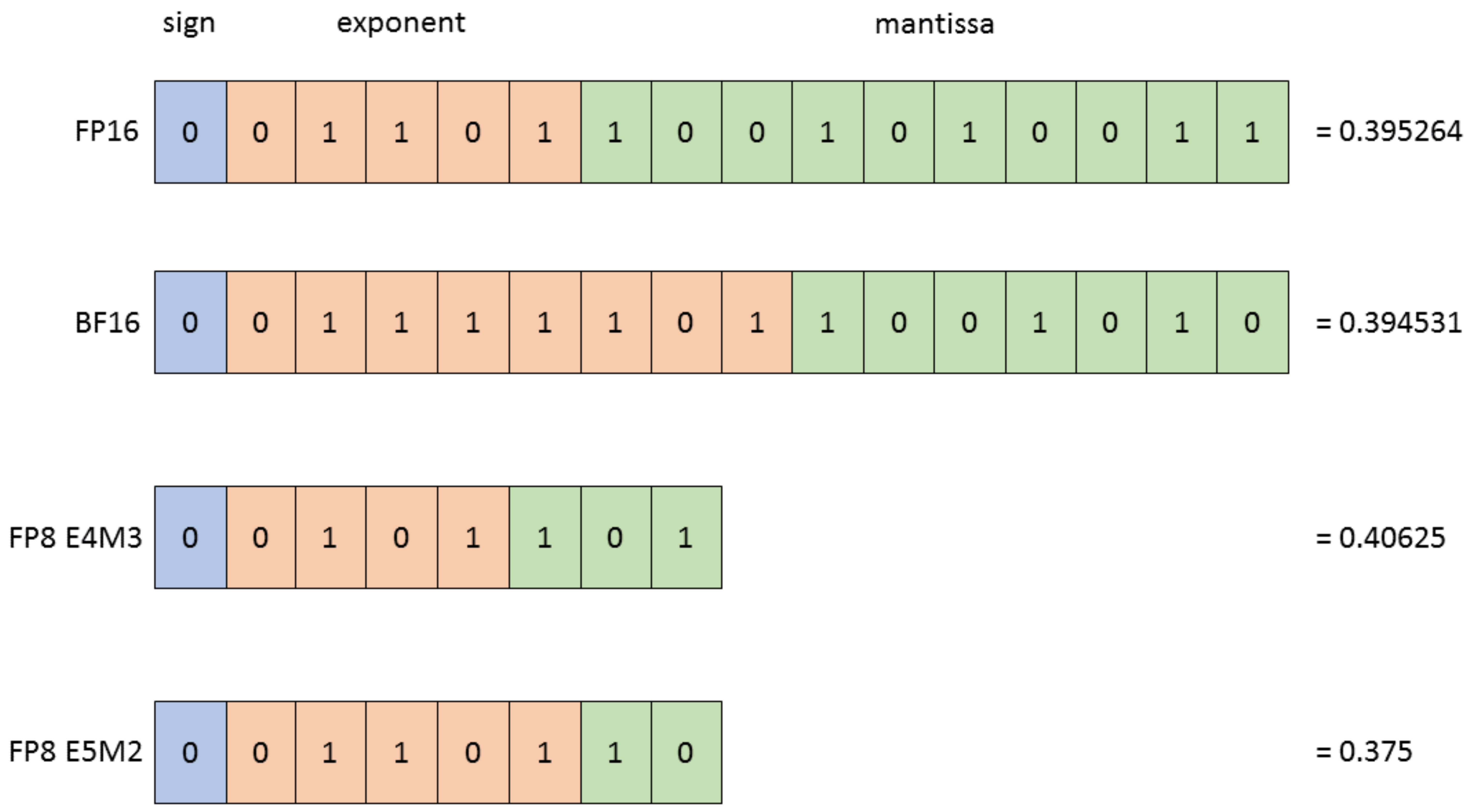
Source: https://arxiv.org/pdf/2106.09685.pdf

Floats



Source: https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/precision.png

Floats



Source: https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/_images/fp8_formats.png

QLora

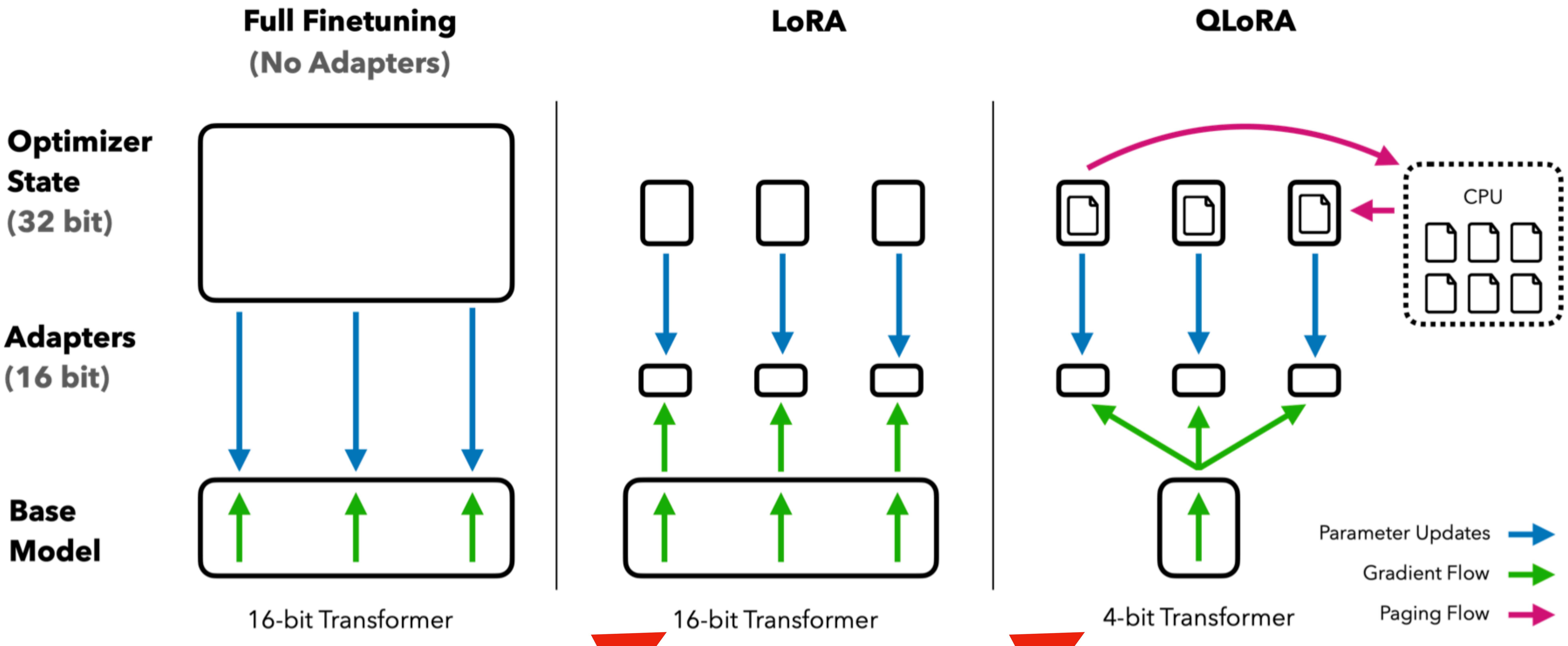


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Source: https://arxiv.org/pdf/2305.14314.pdf

QLora

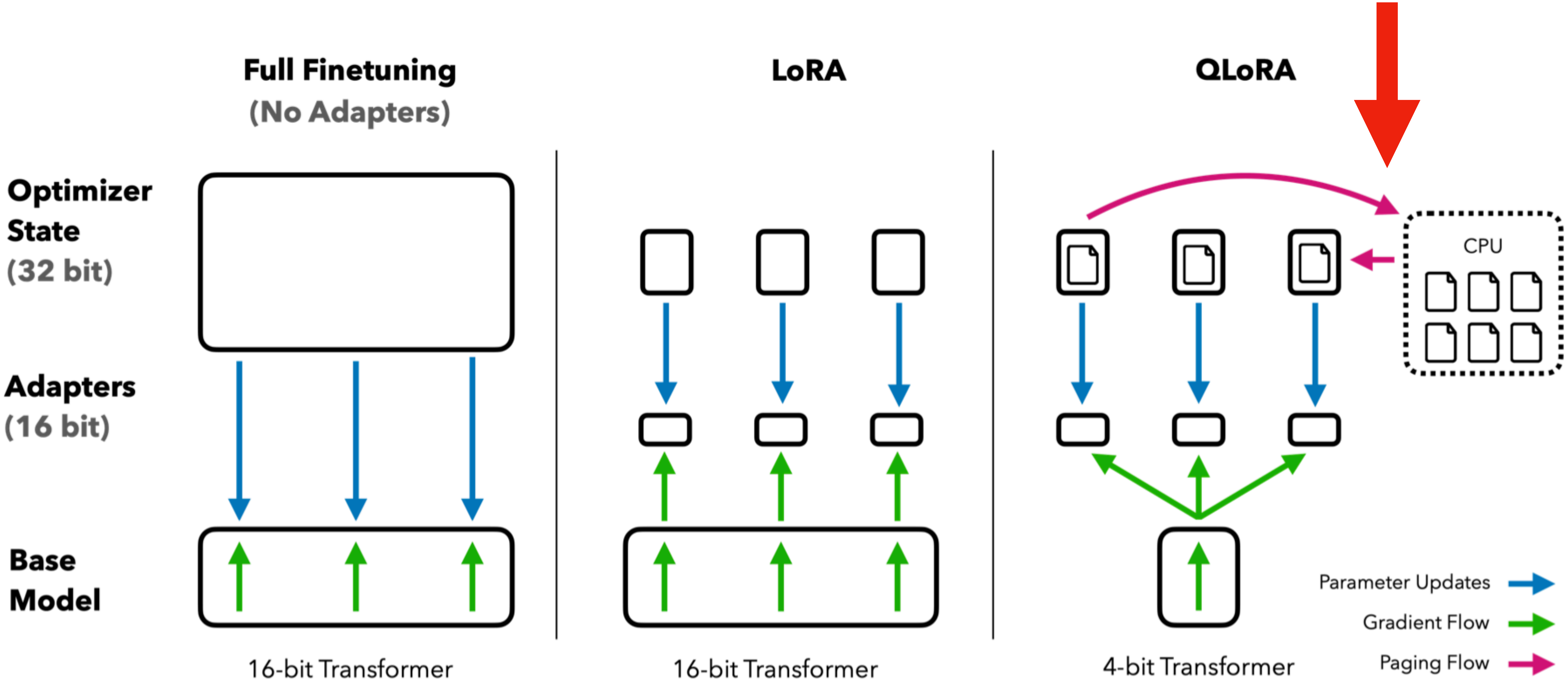


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Source: https://arxiv.org/pdf/2305.14314.pdf

QLora

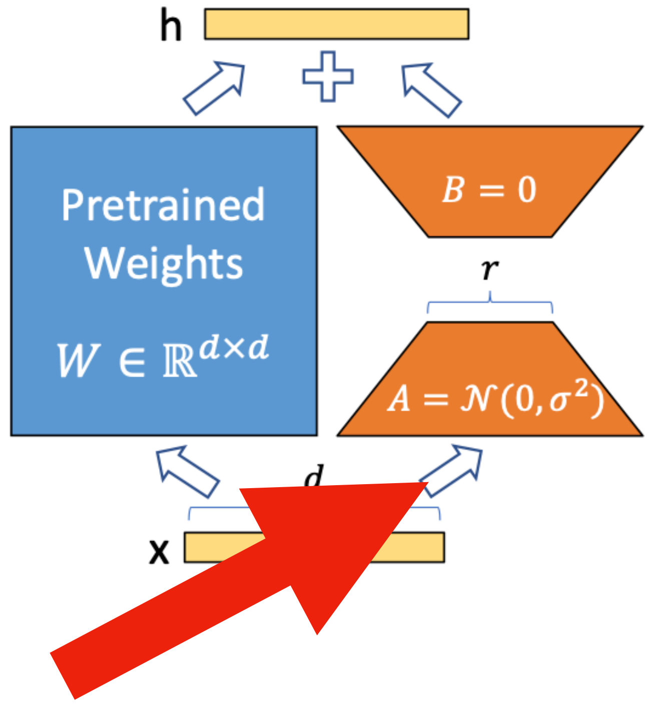
QLORA. Using the components described above, we define QLORA for a single linear layer in the quantized base model with a single LoRA adapter as follows:

$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}}\mathbf{L}_1^{\text{BF16}}\mathbf{L}_2^{\text{BF16}}, \tag{5}$$

where doubleDequant$(\cdot)$ is defined as:

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}}, \tag{6}$$

We use NF4 for $\mathbf{W}$ and FP8 for c_2. We use a blocksize of 64 for $\mathbf{W}$ for higher quantization precision and a blocksize of 256 for c_2 to conserve memory.



Source: https://arxiv.org/pdf/2305.14314.pdf

QLora

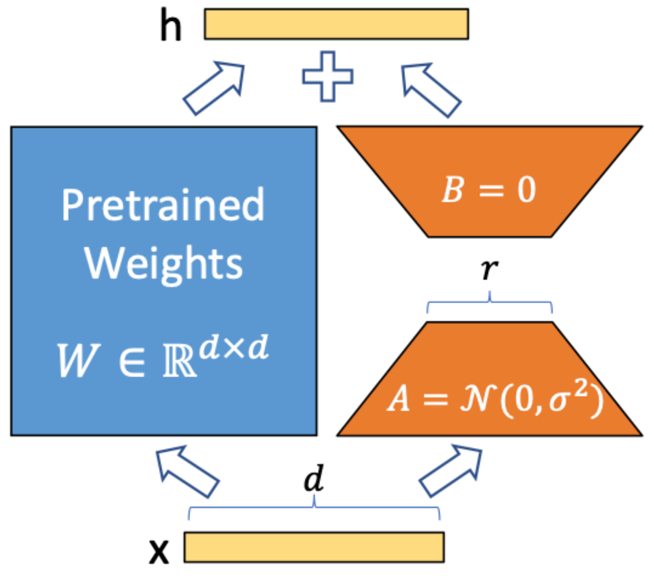
QLoRA. Using the components described above, we define QLoRA for a single linear layer in the quantized base model with a single LoRA adapter as follows:

$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}} \text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}} \mathbf{L}_1^{\text{BF16}} \mathbf{L}_2^{\text{BF16}}, \tag{5}$$

where doubleDequant$(\cdot)$ is defined as:

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}}, \tag{6}$$

We use NF4 for $\mathbf{W}$ and FP8 for c_2. We use a blocksize of 64 for $\mathbf{W}$ for higher quantization precision and a blocksize of 256 for c_2 to conserve memory.



Source: https://arxiv.org/pdf/2305.14314.pdf

Quantization

Block-wise k-bit Quantization Quantization is the process of discretizing an input from a representation that holds more information to a representation with less information. It often means taking a data type with more bits and converting it to fewer bits, for example from 32-bit floats to 8-bit Integers. To ensure that the entire range of the low-bit data type is used, the input data type is commonly rescaled into the target data type range through normalization by the absolute maximum of the input elements, which are usually structured as a tensor. For example, quantizing a 32-bit Floating Point (FP32) tensor into a Int8 tensor with range $[-127, 127]$:

$$\mathbf{X}^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right) = \text{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}), \tag{1}$$

where c is the *quantization constant* or *quantization scale*. Dequantization is the inverse:

$$\text{dequant}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}} \tag{2}$$

Source: https://arxiv.org/pdf/2305.14314.pdf

QLora

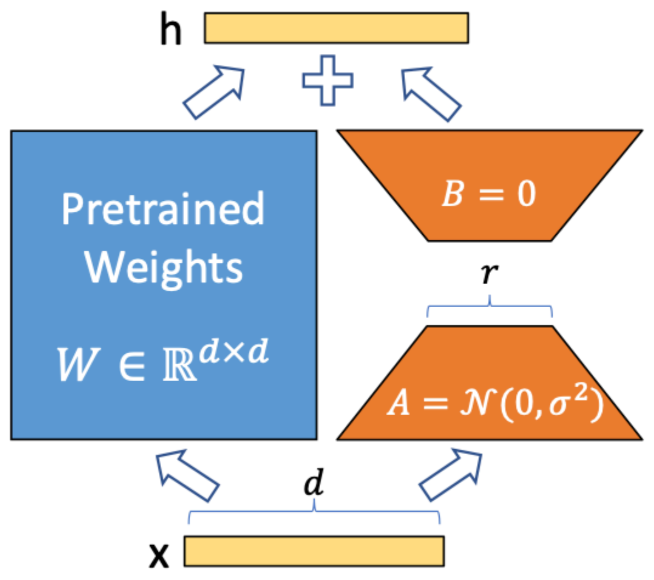
QLoRA. Using the components described above, we define QLoRA for a single linear layer in the quantized base model with a single LoRA adapter as follows:

$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}}\mathbf{L}_1^{\text{BF16}}\mathbf{L}_2^{\text{BF16}}, \tag{5}$$

where doubleDequant$(\cdot)$ is defined as:

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}}, \tag{6}$$

We use NF4 for $\mathbf{W}$ and FP8 for c_2. We use a blocksize of 64 for $\mathbf{W}$ for higher quantization precision and a blocksize of 256 for c_2 to conserve memory.



Source: https://arxiv.org/pdf/2305.14314.pdf

QLora

Table 3: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

Dataset Model	GLUE (Acc.) RoBERTa-large	Super-NaturalInstructions (RougeL) T5-80M	 T5-250M	 T5-780M	 T5-3B	 T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

Source: https://arxiv.org/pdf/2305.14314.pdf

How to use (Q)Lora?

OpenAccess-AI-Collective / **axolotl** Public

README.md

Axolotl

Axolotl is a tool designed to streamline the fine-tuning of various AI models, offering support for multiple configurations and architectures.

Features:

- Train various Huggingface models such as llama, pythia, falcon, mpt
- Supports fullfinetune, lora, qlora, relora, and gptq
- Customize configurations using a simple yaml file or CLI overwrite
- Load different dataset formats, use custom formats, or bring your own tokenized datasets
- Integrated with xformer, flash attention, rope scaling, and multipacking
- Works with single GPU or multiple GPUs via FSDP or Deepspeed
- Easily run with Docker locally or on the cloud
- Log results and optionally checkpoints to wandb
- And more!

Built with Axolotl

axolotl / examples / llama-2 / **qlora.yml**

```yaml
base_model: NousResearch/Llama-2-7b-hf
model_type: LlamaForCausalLM
tokenizer_type: LlamaTokenizer
is_llama_derived_model: true

load_in_8bit: false
load_in_4bit: true
strict: false

datasets:
  - path: mhenrichsen/alpaca_2k_test
    type: alpaca
dataset_prepared_path:
val_set_size: 0.05
output_dir: ./qlora-out

adapter: qlora
lora_model_dir:

sequence_len: 4096
sample_packing: true
pad_to_sequence_len: true
```

```yaml
lora_r: 32
lora_alpha: 16
lora_dropout: 0.05
lora_target_modules:
lora_target_linear: true
lora_fan_in_fan_out:
```

```
accelerate launch -m axolotl.cli.train examples/llama-2/qlora.yml
```