北京大学
PEKING UNIVERSITY

Optimization for Deep Learning (Part III)

Kun Yuan

Center for Machine Learning Research @ Peking University

Fudan University June 8, 2024

北京大学
PEKING UNIVERSITY

PART 01

Fine-Tuning

Finetune

北京大学 PEKING UNIVERSITY

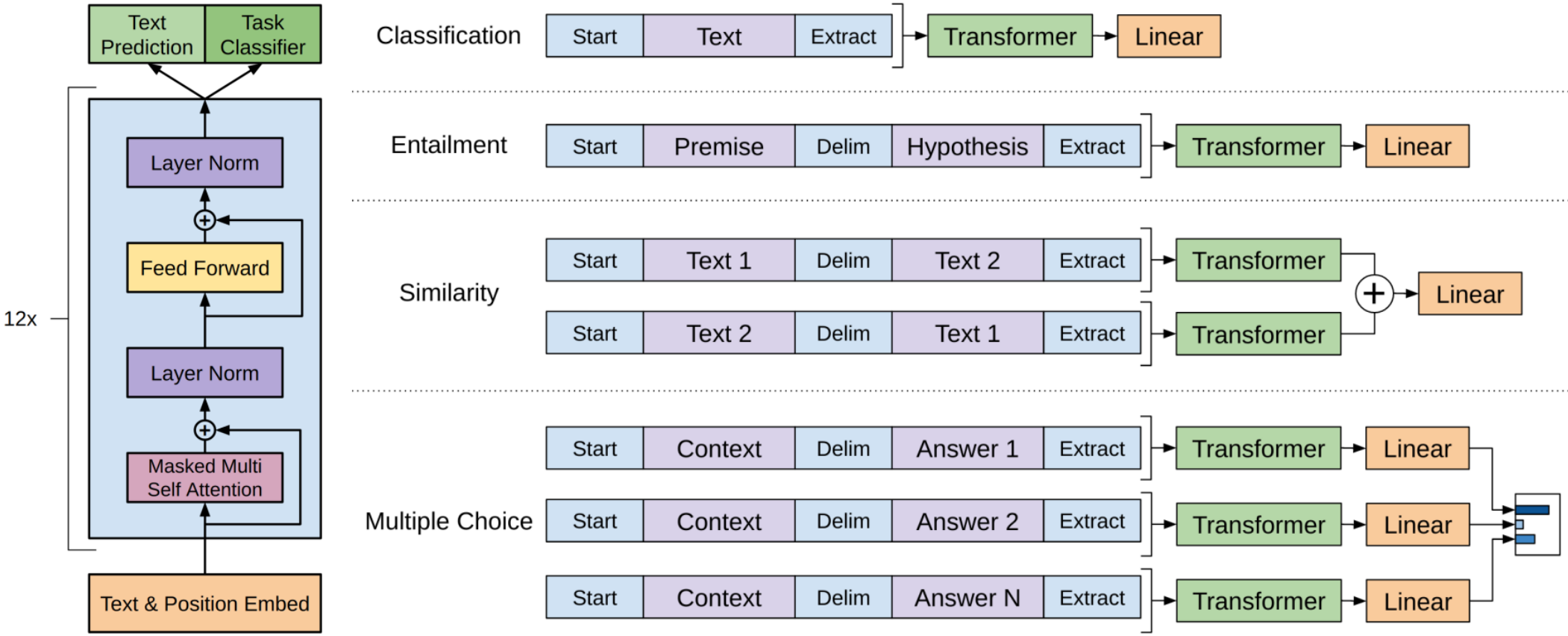
- Fine-tuning in LLMs refers to the process of taking a pre-trained model and further training it on a more specific dataset

- Finetuning will adapt LLM to a particular task or domain. This allows the model to better understand and generate relevant responses for the specific task than pretrained model

- Finetune is an important technique to help you utilize LLMs into your own private applications

GPT finetune

It is very natural to perform fine-tuning with GPT



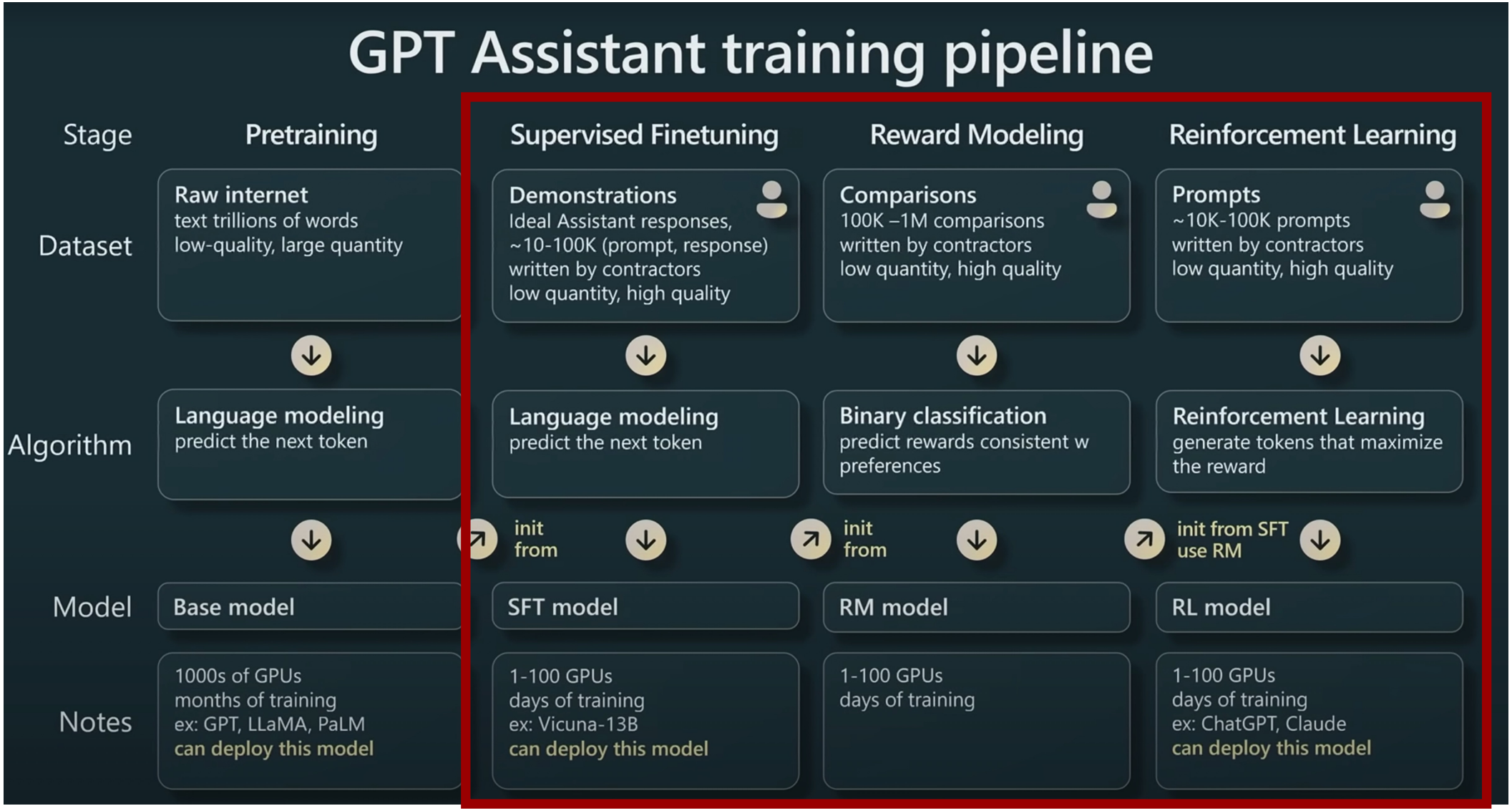
GPT Performance

北京大学 PEKING UNIVERSITY

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

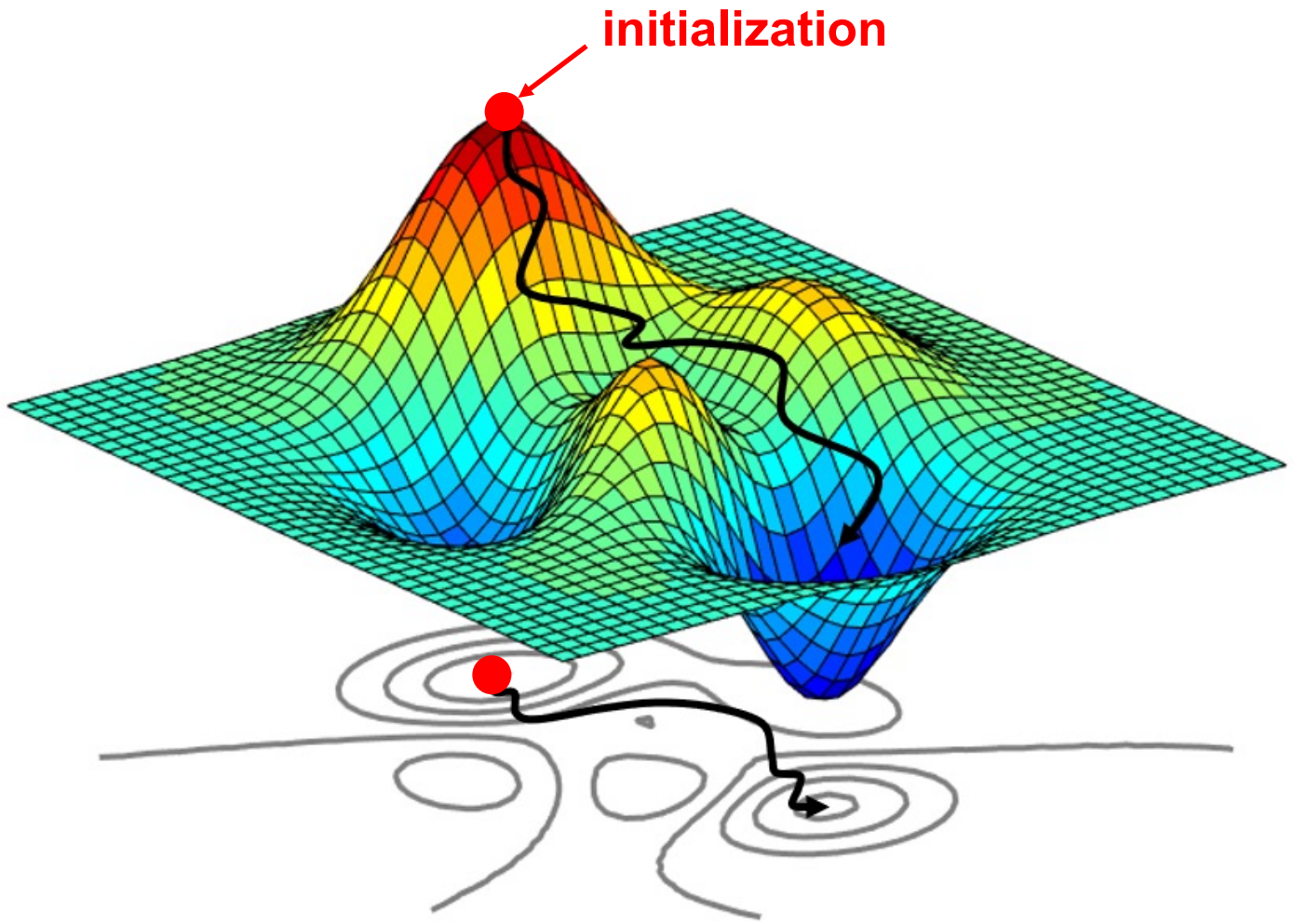
GPT-1 is the first model that significantly improves performances over various tasks

GPT + finetune = ChatGPT



Finetune is essentially retraining the model with nice initialization

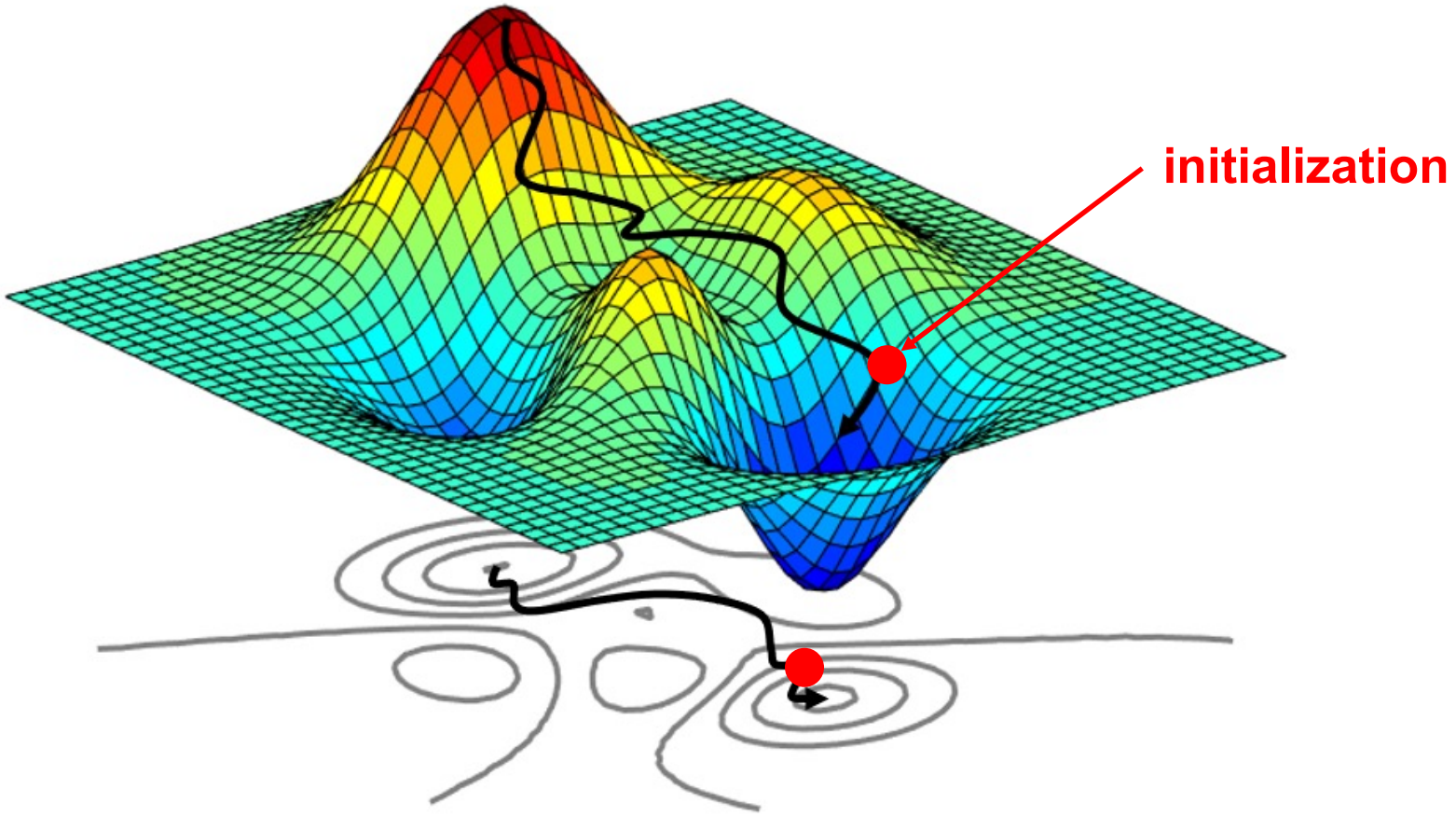
Pretrain



[Image is from a Medium Blog]

Finetune is essentially retraining the model with nice initializations

Finetune



[Image is from a Medium Blog]

Finetuning LLM is very expensive

北京大学 PEKING UNIVERSITY

- **Computational Resources**: Finetuning LLMs, especially the latest ones with billions of parameters like GPT-3 or GPT-4, requires substantial computational power.

- **Data Requirements:** Finetuning an LLM effectively requires a significant amount of high-quality, domain-specific data. Gathering, cleaning, and preparing this data can be labor-intensive and costly.

- **Expertise:** The process of finetuning an LLM requires expertise. Hiring or consulting with experts to handle the finetuning process adds to the overall cost.

- **Maintenance and Experimentation**: Iterative experimentation is often necessary to achieve optimal performance. This means multiple rounds of training, evaluation, and tweaking, each consuming additional resources

Memory-efficient optimizers are in urgent need for fine-tune

北京大学
PEKING UNIVERSITY

PART 02

Low-Rank Fine-tuning

LoRA: Low-Rank Adaptation of LLMs

北京大学 PEKING UNIVERSITY

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

Finetuning LLM with low-rank adaptation

北京大学 PEKING UNIVERSITY

- Full Parameter Fine-tuning:

$$\min_{W \in \mathbb{R}^{p \times q}} \quad \mathbb{E}_{\xi \sim \mathcal{D}}[F(W; \xi)]$$

which can be rewritten into

$$\min_{\Delta W \in \mathbb{R}^{p \times q}} \quad \mathbb{E}_{\xi \sim \mathcal{D}}[F(W + \Delta W; \xi)]$$

- Since ΔW is observed to have low rank, we can assume $\Delta W = AB$ and achieve:

$$\min_{A \in \mathbb{R}^{p \times r}, B \in \mathbb{R}^{r \times q}} \quad \mathbb{E}_{\xi \sim \mathcal{D}}[F(W + AB; \xi)]$$

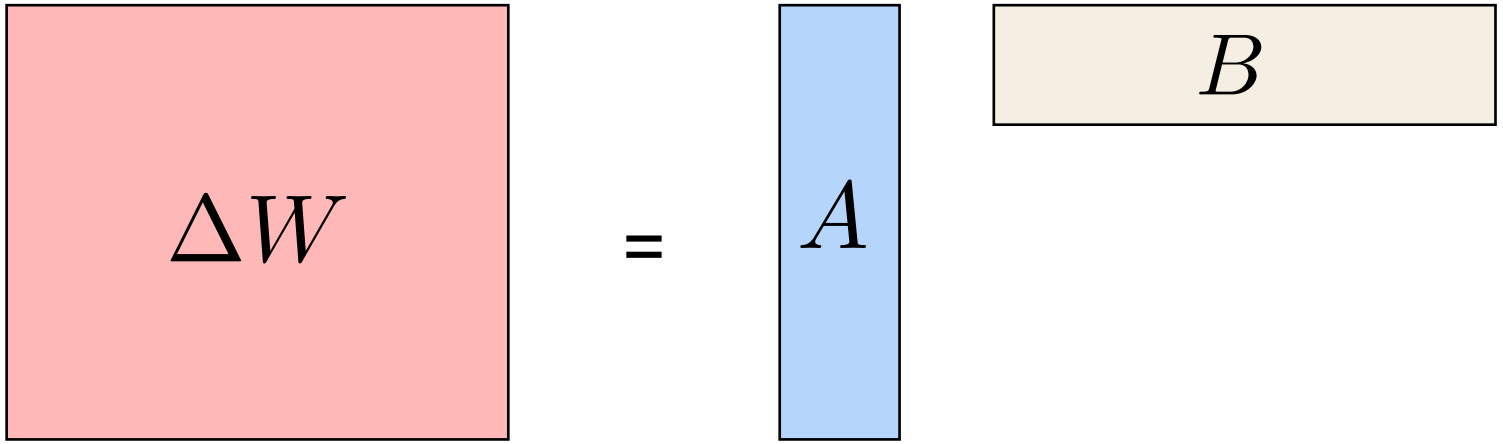
New knowledge and expertise are imposed through low rank matrices A and B

Finetuning LLM with low-rank adaptation

- Since ΔW is observed to have low rank, we can assume $\Delta W = AB$ and achieve:

$$\min_{A \in \mathbb{R}^{p \times r}, B \in \mathbb{R}^{r \times q}} \mathbb{E}_{\xi \sim \mathcal{D}}[F(W + AB; \xi)]$$

New knowledge and expertise are imposed through low rank matrices A and B

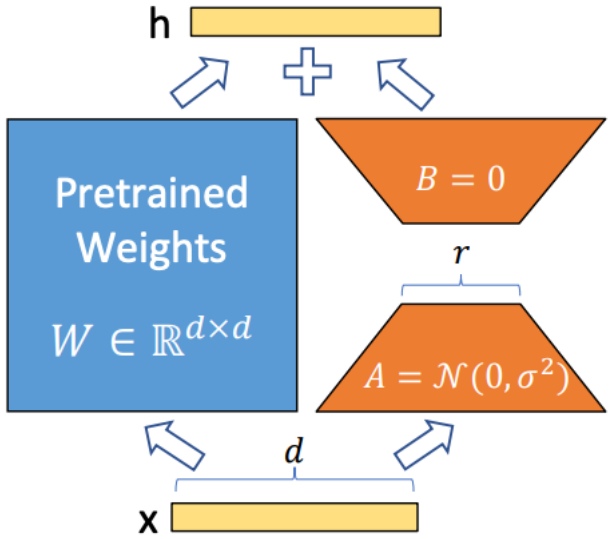


Finetuning LLM with low-rank adaptation

- Finetuning with low-rank adaptation:

$$\min_{A\in\mathbb{R}^{p\times r},B\in\mathbb{R}^{r\times q}} \mathbb{E}_{\xi\sim\mathcal{D}}[F(W+AB;\xi)]$$

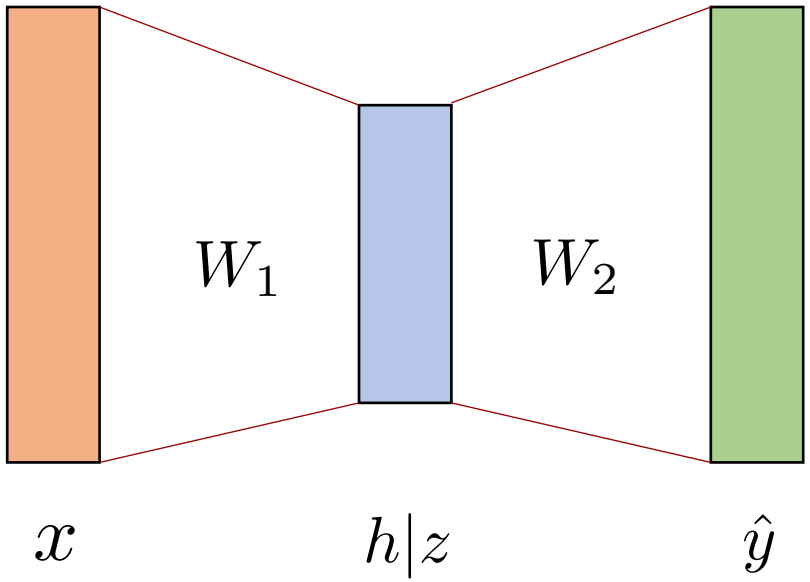
Only A and B are to be trained, while W is fixed



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

$$W'x = (W + AB)x = Wx + ABx$$

Full parameter fine-tuning: Memory cost



dims: d (p,d) p (q,p) q

$$h = W_1 x$$

$$z = \sigma(h)$$

$$\hat{y} = W_2 z$$

$$f = L(\hat{y})$$

Forward

$$\frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial h} x^T$$

$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla \sigma(h)$$

$$\frac{\partial f}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} z^T, \quad \frac{\partial f}{\partial z} = W_2^T \frac{\partial L}{\partial \hat{y}}$$

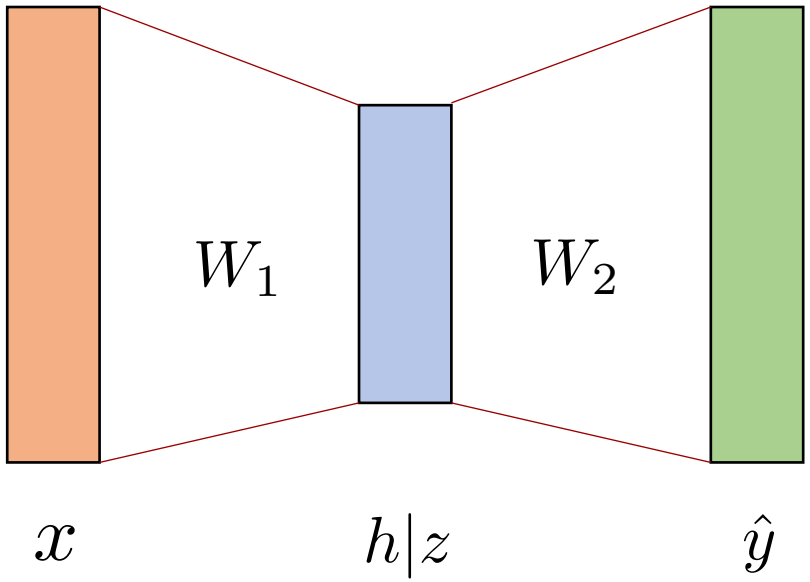
$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

Backward

Store h, z and $\hat{y}$

Store $\nabla_{W_1} f(W_1)$ and $\nabla_{W_2} f(W_2)$

LoRA: Memory cost



dims: d (p,d) p (q,p) q

$$h = (W_1 + A_1B_1)x$$

$$z = \sigma(h)$$

$$\hat{y} = (W_2 + A_2B_2)z$$

$$f = L(\hat{y})$$

Forward

Store h, z and $\hat{y}$

$$\frac{\partial f}{\partial A_1} = \frac{\partial f}{\partial h}(B_1x)^T$$

$$\frac{\partial f}{\partial B_1} = A_1^T\frac{\partial f}{\partial h}(x)^T$$

$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla\sigma(h)$$

$$\frac{\partial f}{\partial z} = (W_2 + A_2B_2)^T\frac{\partial f}{\partial \hat{y}}$$

$$\frac{\partial f}{\partial A_2} = \frac{\partial f}{\partial \hat{y}}(B_2z)^T$$

$$\frac{\partial f}{\partial B_2} = A_2^T\frac{\partial f}{\partial \hat{y}}(z)^T$$

$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

Store $\nabla_{A_1}f$, $\nabla_{B_1}f$ and $\nabla_{A_2}f$, $\nabla_{B_2}f$

How much memory can LoRA save?

- Memory cost:

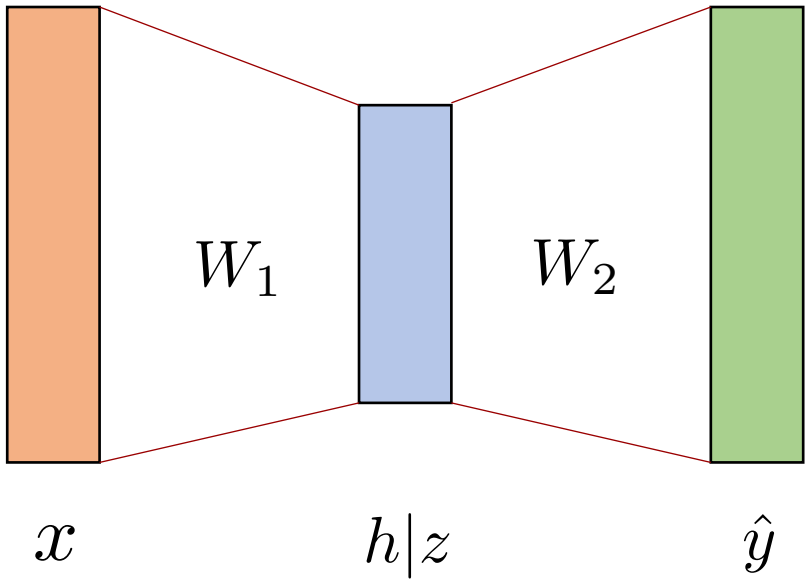
$$\text{Memory cost} = \text{Models} + \text{Grads} + \text{Optimizers} + \text{Activations}$$

- LoRA does **not save activation-incurred memory**; does not apply to scenarios where activation-incurred memory dominates, e.g., long-context transformers

- LoRA **saves models/grads/optimizers** from O(p*q) to O((p+q)*r)

- LoRA can save great memory when rank r is small, and activation-incurred memory is trivial

Full parameter fine-tuning：Computational cost



dims: d (p,d) p (q,p) q

$$h = W_1 x$$

$$z = \sigma(h)$$

$$\hat{y} = W_2 z$$

$$f = L(\hat{y})$$

Forward

pd+qp

$$\frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial h} x^T$$

$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla \sigma(h)$$

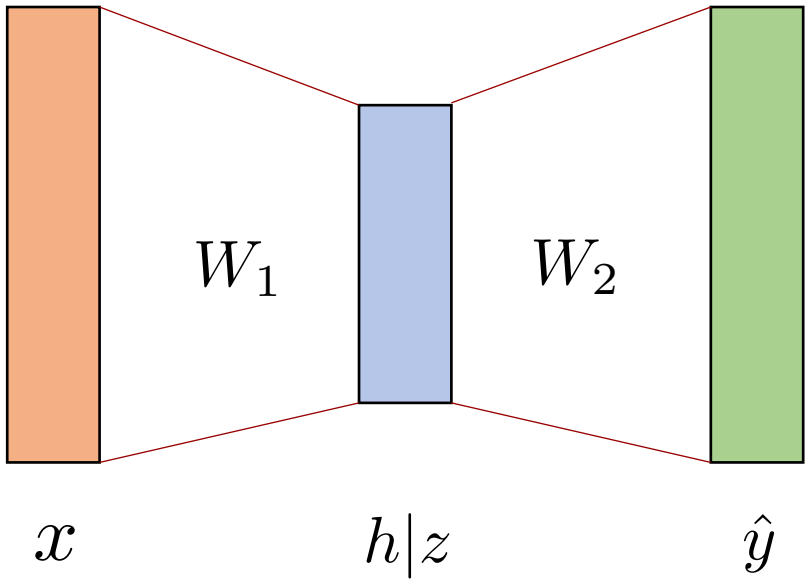
$$\frac{\partial f}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} z^T, \quad \frac{\partial f}{\partial z} = W_2^{\mathsf{T}} \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

Backward

2(pd+qp) + 2pd

LoRA: Computational cost



dims: d (p,d) p (q,p) q

Does not save computations!

$$\mathrm{h} = (W_1 + A_1B_1)x$$

$$z = \sigma(h)$$

$$\hat{y} = (W_2 + A_2B_2)z$$

$$f = L(\hat{y})$$

Forward

$$\frac{\partial f}{\partial A_1} = \frac{\partial f}{\partial h}(B_1x)^T$$

$$\frac{\partial f}{\partial B_1} = A_1^T\frac{\partial f}{\partial h}(x)^T$$

$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla\sigma(h)$$

$$\frac{\partial f}{\partial z} = (W_2 + A_2B_2)^T\frac{\partial f}{\partial \hat{y}}$$

$$\frac{\partial f}{\partial A_2} = \frac{\partial f}{\partial \hat{y}}(B_2z)^T$$

$$\frac{\partial f}{\partial B_2} = A_2^T\frac{\partial f}{\partial \hat{y}}(z)^T$$

$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

pd+qp+dr+pr+pr+qr **3(pr+qr) + 2(pr+dr) + pq**

Results

北京大学 PEKING UNIVERSITY

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

LoRA+

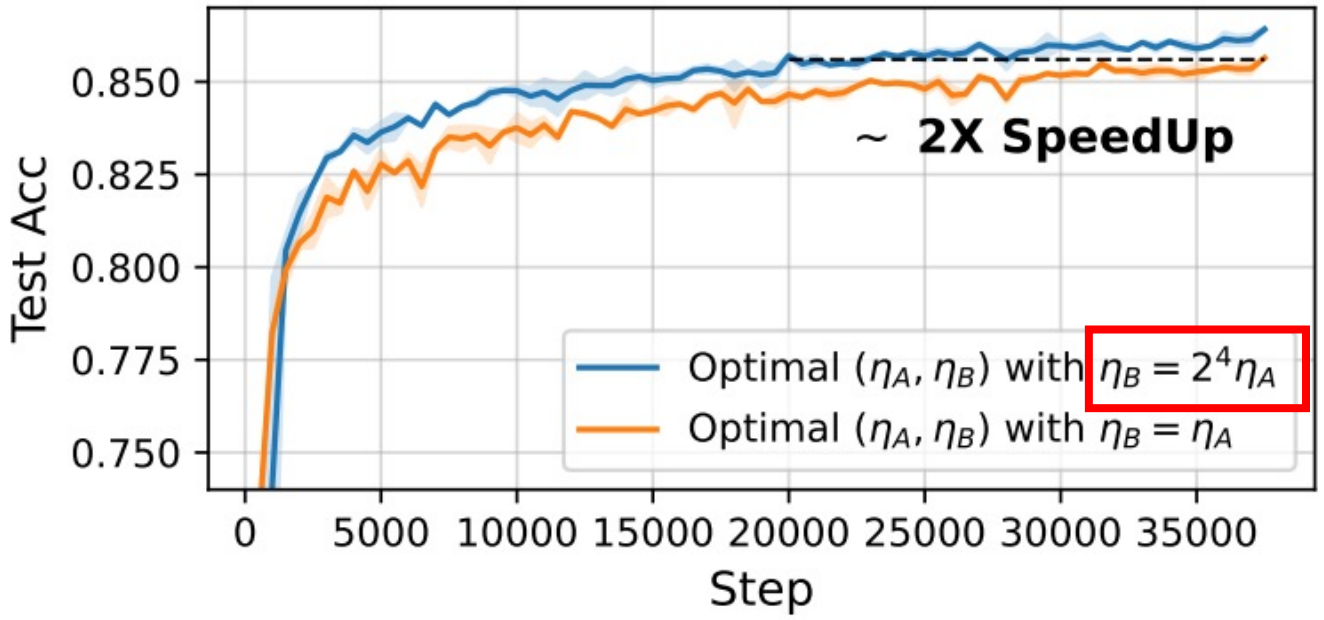
- Since one of the matrices A or B will be initialized to 0, it will be very slow for it to update
- We should use different learning rates for A and B; LoRA+

	LoRA	LoRA+
Parameterization	Pretrained Weights $W \in \mathbb{R}^{n \times n}$ $+ B \times A$	
Training	$A \leftarrow A - \eta \times G_A$ $B \leftarrow B - \eta \times G_B$	$A \leftarrow A - \eta \times G_A$ $B \leftarrow B - \lambda\eta \times G_B$ $\lambda \gg 1$

[LoRA+: Efficient Low Rank Adaptation of Large Models]

LoRA+

- We should use different learning rates for A and B; LoRA+

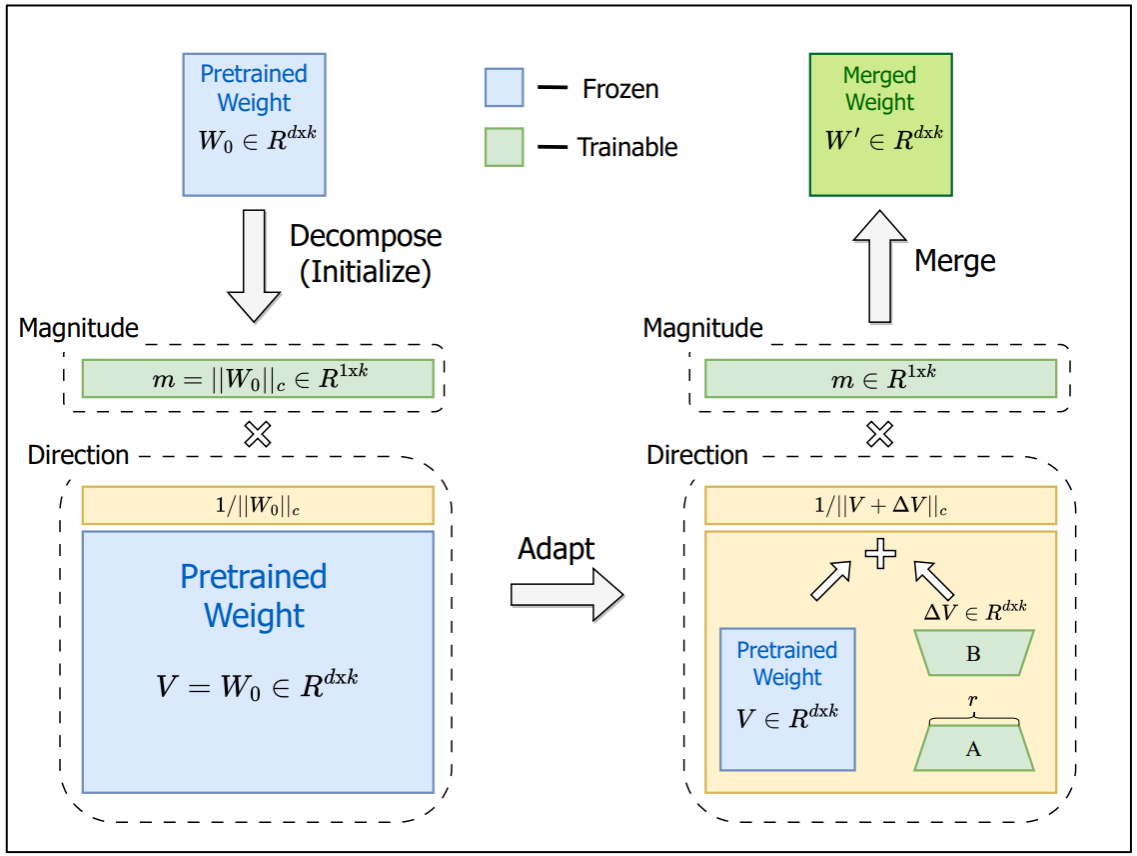


[LoRA+: Efficient Low Rank Adaptation of Large Models]

Weight-Decomposed Low-Rank Adaptation (DoRA)

- DoRA: train the magnitudes and directions separately

$$W' = m\frac{V+\Delta V}{||V+\Delta V||_c} = m\frac{W_0+\underline{BA}}{||W_0+\underline{BA}||_c}$$



Weight-Decomposed Low-Rank Adaptation (DoRA)

北京大学 PEKING UNIVERSITY

Model	PEFT Method	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	Prefix	0.11	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Series	0.99	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel	3.54	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	LoRA	0.83	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA† (Ours)	0.43	70.0	82.6	79.7	83.2	80.6	80.6	65.4	77.6	**77.5**
	DoRA (Ours)	0.84	69.7	83.4	78.6	87.2	81.0	81.9	66.2	79.2	**78.4**
LLaMA-13B	Prefix	0.03	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	Series	0.80	71.8	83	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Parallel	2.89	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.4
	LoRA	0.67	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA† (Ours)	0.35	72.5	85.3	79.9	90.1	82.9	82.7	69.7	83.6	**80.8**
	DoRA (Ours)	0.68	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	**81.5**
LLaMA2-7B	LoRA	0.83	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA† (Ours)	0.43	72.0	83.1	79.9	89.1	83.0	84.5	71.0	81.2	**80.5**
	DoRA (Ours)	0.84	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	**79.7**
LLaMA3-8B	LoRA	0.70	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	DoRA† (Ours)	0.35	74.5	88.8	80.3	95.5	84.7	90.1	79.1	87.2	**85.0**
	DoRA (Ours)	0.71	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	**85.2**

北京大学
PEKING UNIVERSITY

PART 03

Layer-wise fine-tuning (block coordinate minimization)

LISA: Layer-wise finetuning

北京大学 PEKING UNIVERSITY

- LoRA does not update the whole model. Instead, it only update a low-rank incremental model

$$\min_{A\in\mathbb{R}^{p\times r},B\in\mathbb{R}^{r\times q}} \mathbb{E}_{\xi\sim\mathcal{D}}[F(W+AB;\xi)]$$

- Can we update the full parameters in a memory-efficient manner? This can significantly increase the learnable parameters, which is expected to lead to superior performance

- Main idea: Layer-wise finetuning

$$\min_{W} \mathbb{E}_{\xi\sim\mathcal{D}}[F(W;\xi)] = \mathbb{E}_{\xi\sim\mathcal{D}}[F(W_1, W_2, \cdots, W_L;\xi)]$$

where $W := \{W_1, W_2, \cdots, W_L\}$. When update W_ℓ, the other layers remain freeze

[LISA: Layerwise Importance Sampling for Memory-Efficient Large Language Model Fine-Tuning]

LISA: Layer-wise finetuning

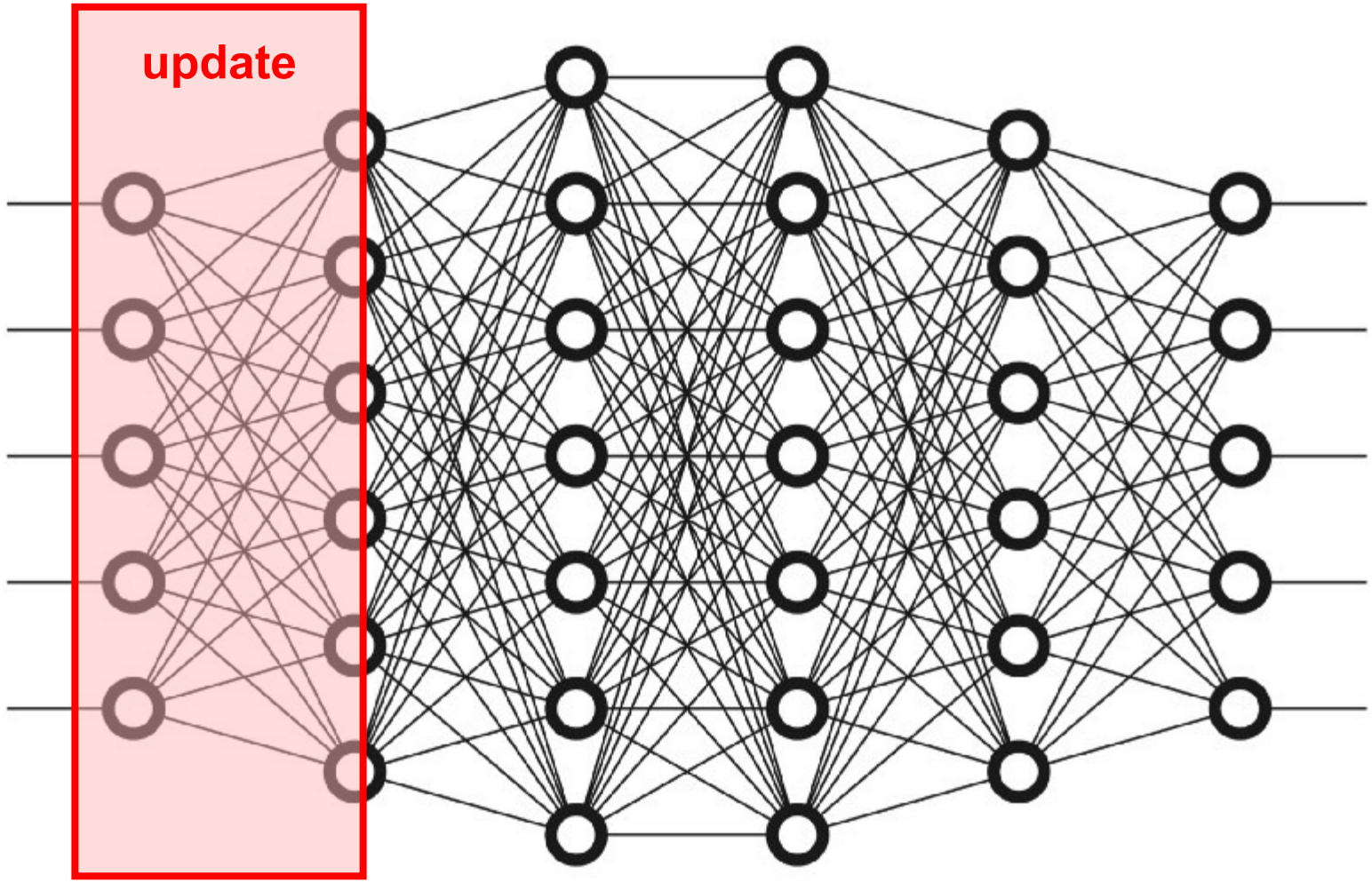
Algorithm 1 Layerwise Importance Sampling AdamW (**LISA**)

Require: number of layers N_L, number of iterations T, sampling period K, number of sampled layers γ, initial learning rate η_0

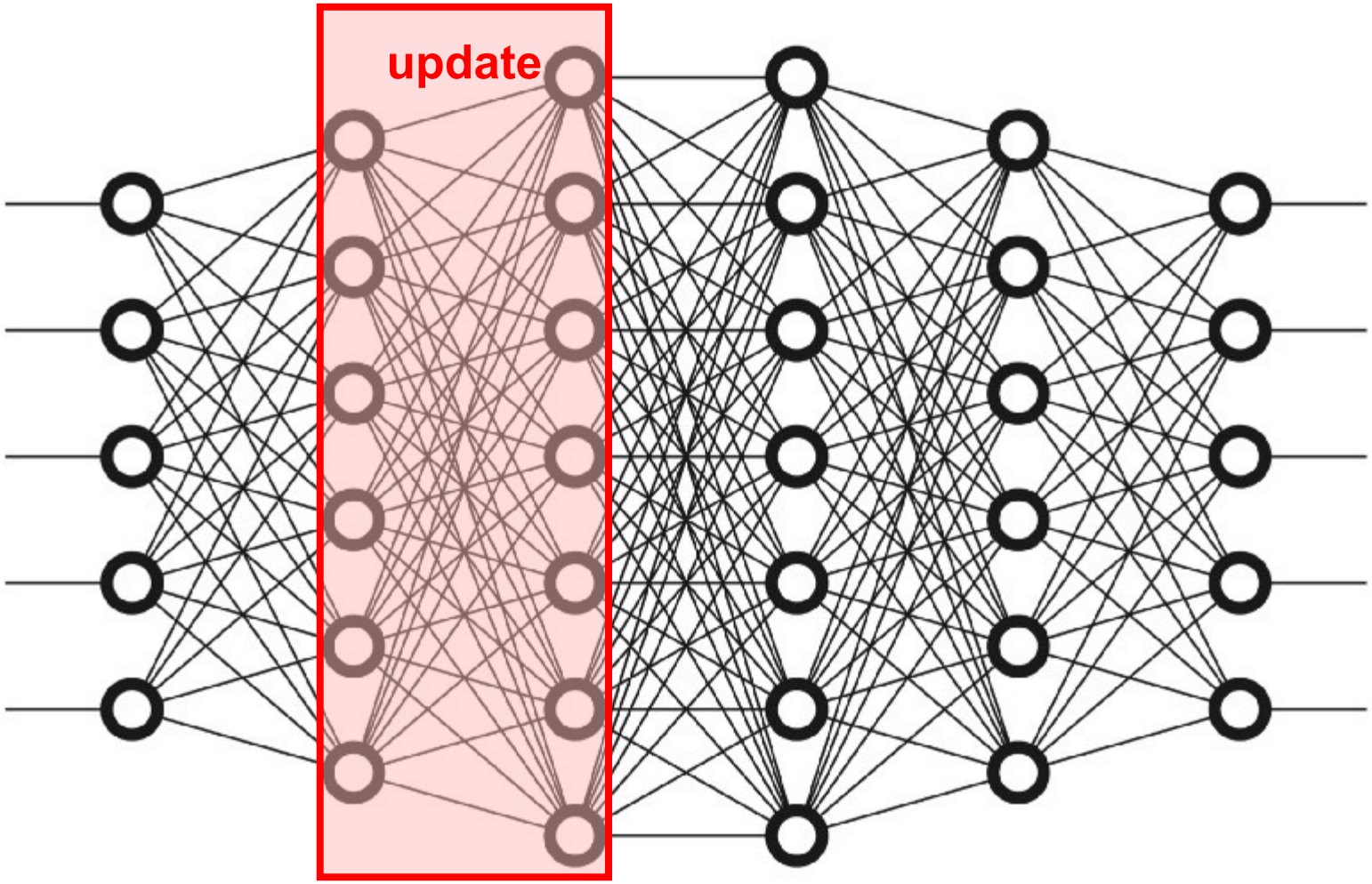
1: **for** $i \leftarrow 0$ to $T/K - 1$ **do**
2: Freeze all layers except the embedding and language modeling head layer
3: Randomly sample γ intermediate layers to unfreeze
4: Run AdamW for K iterations with $\{\eta_t\}_{t=ik}^{ik+k-1}$
5: **end for**

This is basically block coordinate minimization

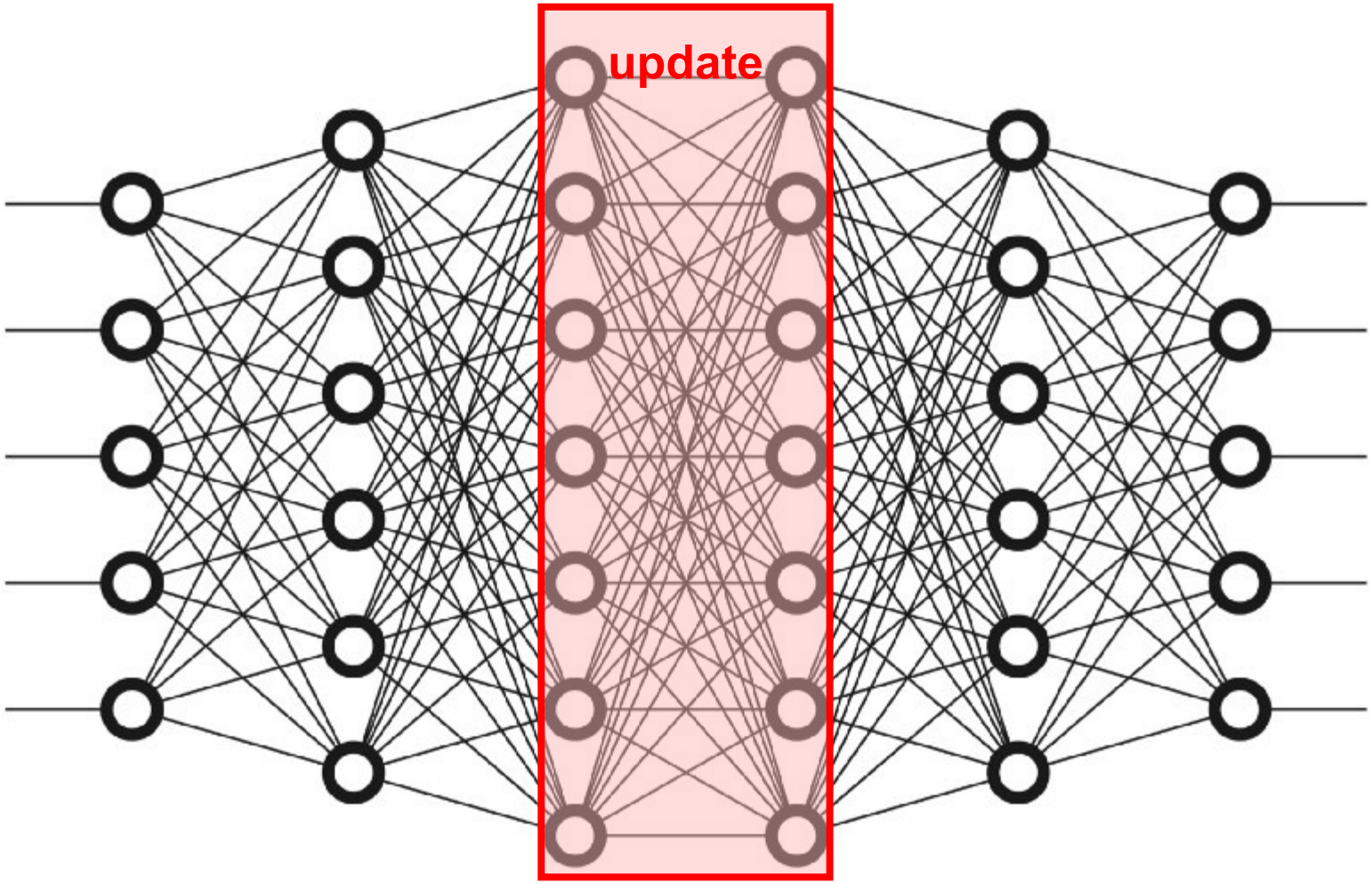
LISA: Layer-wise finetuning



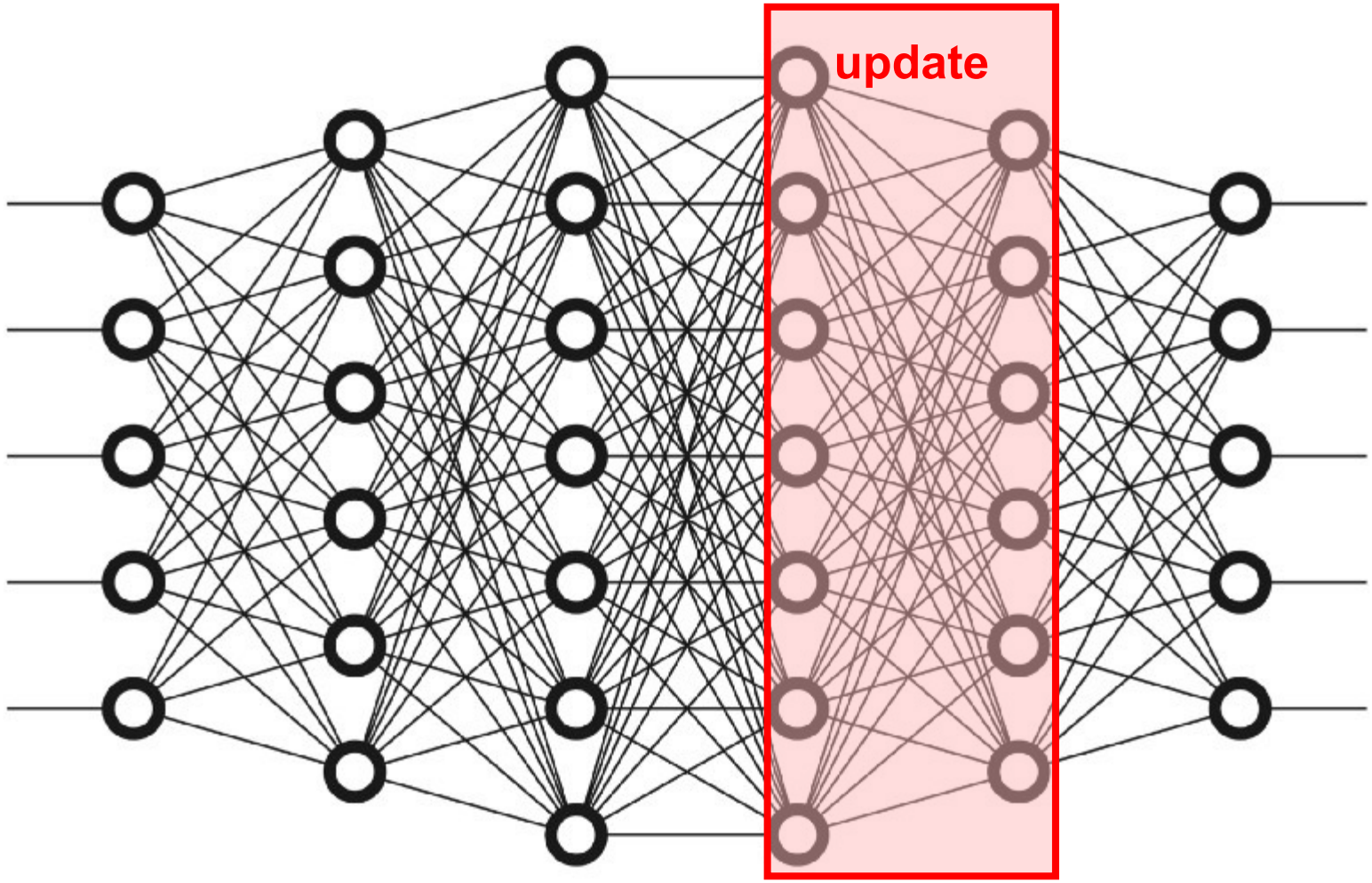
LISA: Layer-wise finetuning



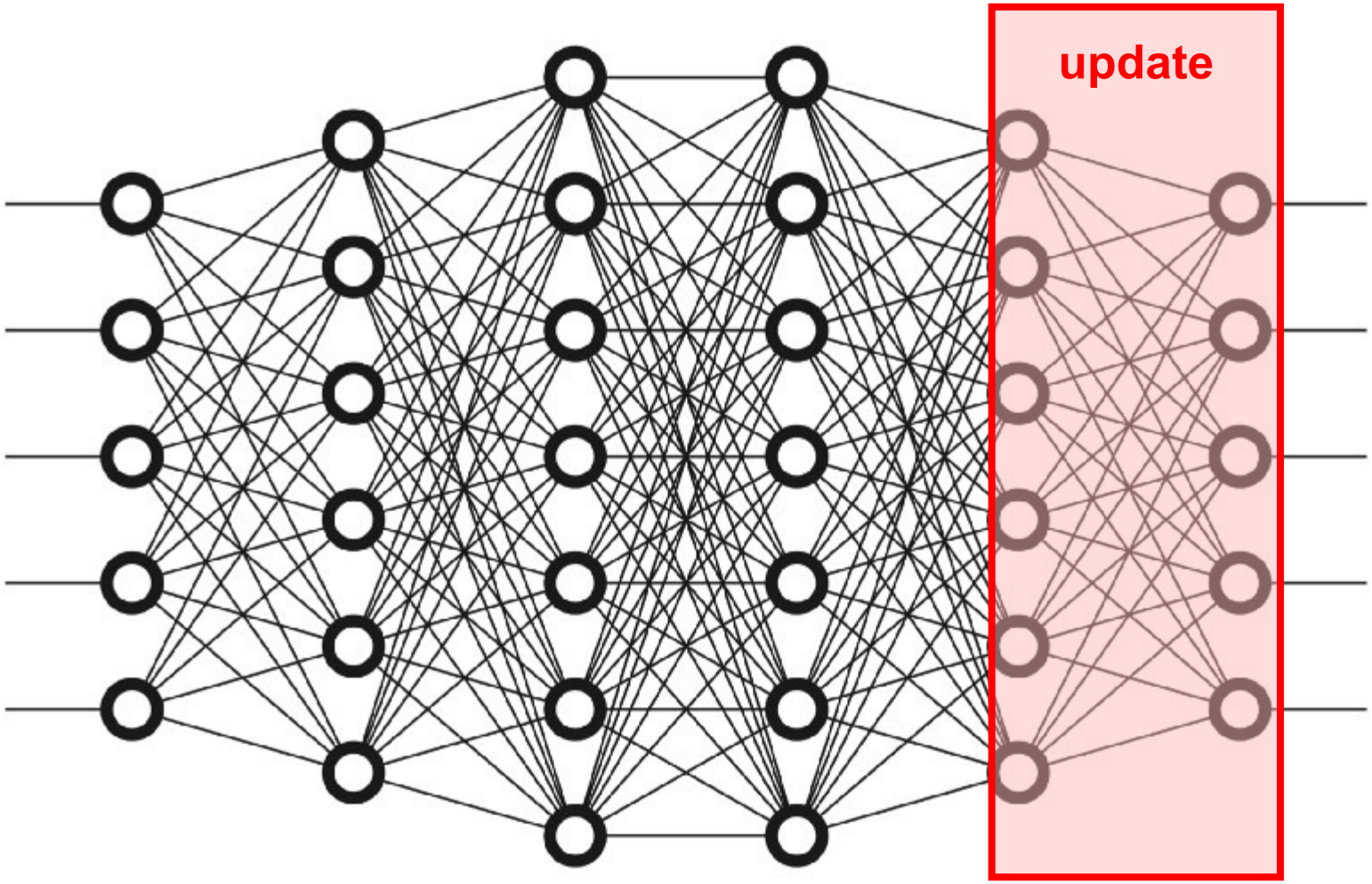
LISA: Layer-wise finetuning



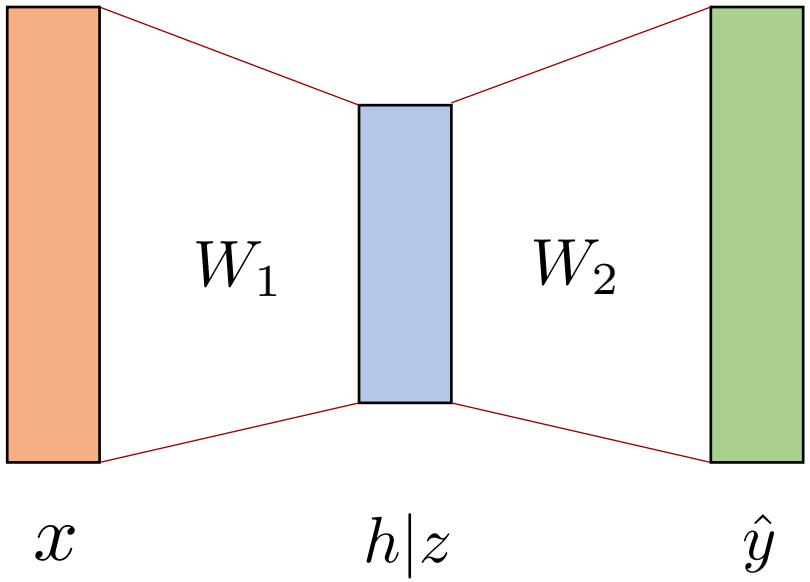
LISA: Layer-wise finetuning



LISA: Layer-wise finetuning



Full parameter fine-tuning: Memory cost



dims: d (p,d) p (q,p) q

$$h = W_1 x$$

$$z = \sigma(h)$$

$$\hat{y} = W_2 z$$

$$f = L(\hat{y})$$

Forward

$$\frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial h} x^T$$

$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla \sigma(h)$$

$$\frac{\partial f}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} z^T, \quad \frac{\partial f}{\partial z} = W_2^T \frac{\partial L}{\partial \hat{y}}$$

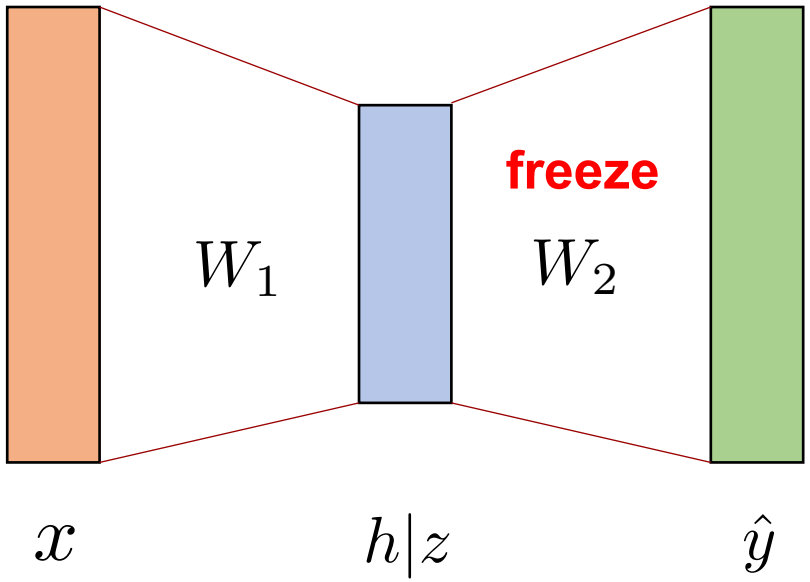
$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

Backward

Store h, z and $\hat{y}$

Store $\nabla_{W_1} f(W_1)$ and $\nabla_{W_2} f(W_2)$

LISA memory analysis



dims: d (p,d) p (q,p) q

Save activations

$$h = W_1 x$$

$$z = \sigma(h)$$

$$\hat{y} = W_2 z$$

$$f = L(\hat{y})$$

Forward

Store h and $\hat{y}$

$$\frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial h} x^T$$

$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla\sigma(h)$$

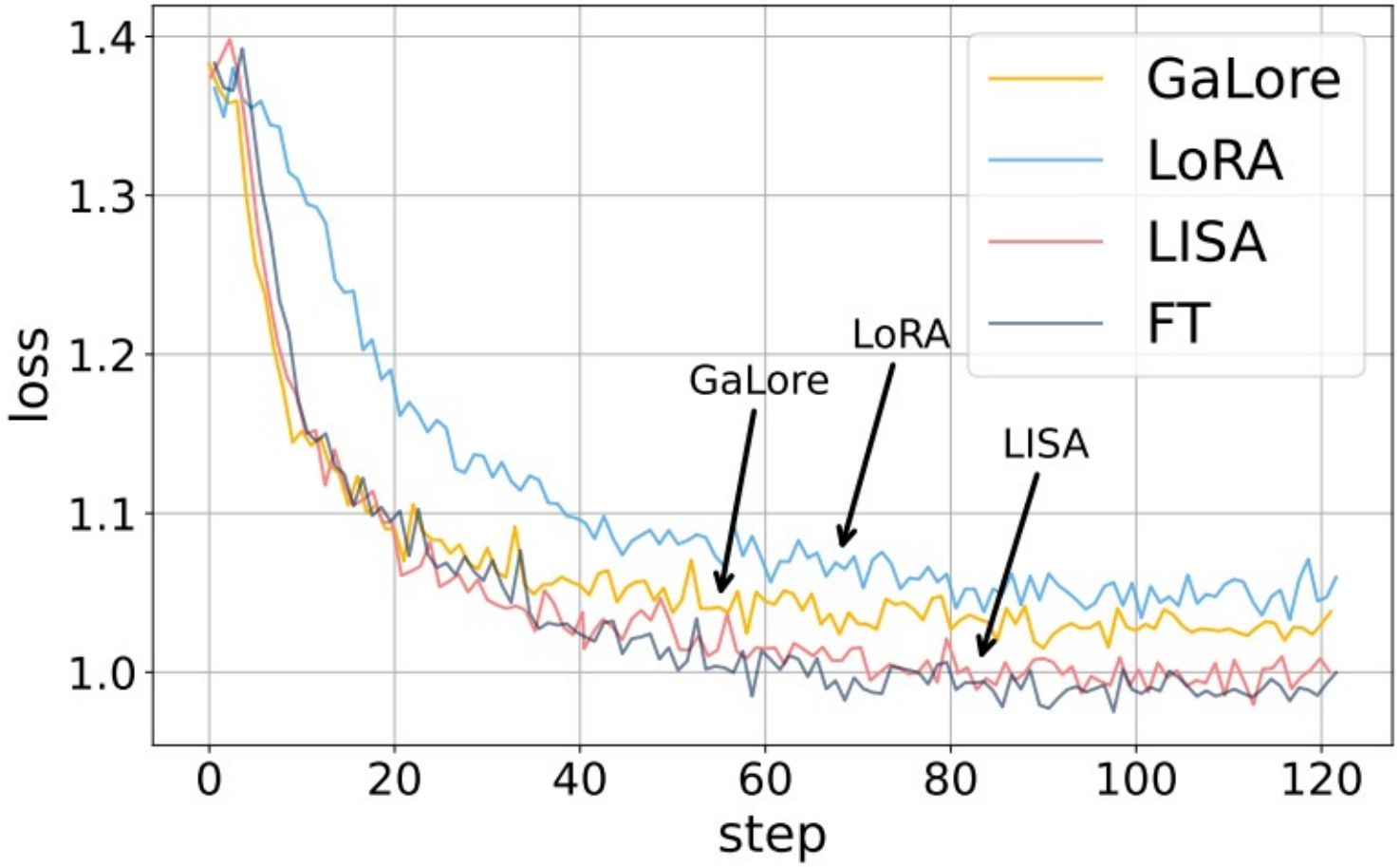
$$\frac{\partial f}{\partial z} = W_2^T \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

Backward

Store $\nabla_{W_1} f(W_1)$

LISA performance



LISA performance

Table 2: Results of different methods on MMLU, AGIEval, and WinoGrande, measured by accuracy.

Model	Method	MMLU (5-shot)	AGIEval (3-shot)	WinoGrande (5-shot)
TinyLlama	Vanilla	25.50	19.55	59.91
	LoRA	25.81	19.82	61.33
	GaLore	25.21	21.19	61.09
	LISA	**26.02**	**21.71**	61.48
	FT	25.62	21.28	**62.12**
Mistral-7B	Vanilla	60.12	26.79	79.24
	LoRA	61.78	27.56	78.85
	GaLore	57.87	26.23	75.85
	LISA	**62.09**	**29.76**	**78.93**
	FT	61.70	28.07	78.85
LLaMA-2-7B	Vanilla	45.87	25.69	74.11
	LoRA	45.50	24.73	74.74
	GaLore	45.56	24.39	73.32
	LISA	**46.21**	26.06	**75.30**
	FT	45.66	**27.02**	75.06

北京大学
PEKING UNIVERSITY

PART 04

Zeroth-order fine-tuning

Zeroth-order Finetuning

- Consider the unconstrained optimization problem

$$\min_{x \in \mathbb{R}^d} \quad f(x)$$

- Suppose computing $\nabla f(x)$ is very expensive, we can estimate the gradient with finite difference

Let $f(x) : \mathbb{R}^d \to \mathbb{R}$, we have

$$\frac{\partial f(x)}{\partial x_i} = \lim_{\tau \to 0} \frac{f(x + \tau e_i) - f(x)}{\tau}, \quad i = 1, 2, \ldots, d.$$

where e_i is the i-th column of the identity matrix I.

- The gradient $\nabla f(x) \in \mathbb{R}^d$ is defined as $\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_d} \right)^\top = \sum_{i=1}^{d} \frac{\partial f(x)}{\partial x_i} e_i$

Zeroth-order Finetuning

- Consider the unconstrained optimization problem

$$\min_{x \in \mathbb{R}^d} \quad f(x)$$

- Zeroth-order gradient descent is given as follows

$$g(x_k) = \sum_{i=1}^{d} \frac{f(x_k + \tau e_i) - f(x_k)}{\tau} e_i$$

$$x_{k+1} = x_k - \gamma g(x_k)$$

- Needs d+1 queries to estimate one gradient

Zeroth-order Finetuning

- Consider the unconstrained optimization problem

$$\min_{x \in \mathbb{R}^d} \quad f(x)$$

- Zeroth-order gradient descent with **Gaussian sampling**

$$g(x_k) = \frac{f(x_k + \tau z) - f(x_k)}{\tau} z \qquad z \sim \mathcal{N}(0, I_d)$$

$$x_{k+1} = x_k - \gamma g(x_k)$$

- Needs 2 queries to estimate one gradient

Zeroth-order Finetuning

- Consider the unconstrained stochastic optimization problem

$$x^\star = \arg\min_{x \in \mathbb{R}^d} \{\mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{D}}[F(x; \boldsymbol{\xi})]\}$$

- Zeroth-order **stochastic** gradient descent with **Gaussian sampling**

$$g(x_k) = \frac{F(x_k + \tau z; \xi_k) - F(x_k; \xi_k)}{\tau} z \qquad z \sim \mathcal{N}(0, I_d)$$

$$x_{k+1} = x_k - \gamma g(x_k)$$

- Needs 2 queries to estimate one gradient

Zeroth-order Finetuning

北京大学 PEKING UNIVERSITY

- Recall the memory cost in when training DNN with Adam:

$$\text{Memory} = \text{Model} + \text{Gradient} + \text{Optimizer states} + \text{Activations}$$

- If we train DNN with zeroth-order SGD

$$\text{Memory} = \text{Model} + \text{Gradient} + \text{Optimizer states} + \text{Activations}$$

Zeroth-order Finetuning

Algorithm 1: MeZO

Require: parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, step budget T, perturbation scale ϵ, batch size B, learning rate schedule $\{\eta_t\}$

for $t = 1, ..., T$ **do**
- Sample batch $\mathcal{B} \subset \mathcal{D}$ and random seed s
- $\boldsymbol{\theta} \leftarrow$ `PerturbParameters`$(\boldsymbol{\theta}, \epsilon, s)$
- $\ell_+ \leftarrow \mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$
- $\boldsymbol{\theta} \leftarrow$ `PerturbParameters`$(\boldsymbol{\theta}, -2\epsilon, s)$
- $\ell_- \leftarrow \mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$
- $\boldsymbol{\theta} \leftarrow$ `PerturbParameters`$(\boldsymbol{\theta}, \epsilon, s)$ ▷ `Reset parameters before descent`
- `projected_grad` $\leftarrow (\ell_+ - \ell_-)/(2\epsilon)$
- Reset random number generator with seed s ▷ `For sampling` z
- **for** $\theta_i \in \boldsymbol{\theta}$ **do**
  - $z \sim \mathcal{N}(0, 1)$
  - $\theta_i \leftarrow \theta_i - \eta_t *$ `projected_grad` $* z$
- **end**

end

Subroutine `PerturbParameters`$(\boldsymbol{\theta}, \epsilon, s)$
- Reset random number generator with seed s ▷ `For sampling` z
- **for** $\theta_i \in \boldsymbol{\theta}$ **do**
  - $z \sim \mathcal{N}(0, 1)$
  - $\theta_i \leftarrow \theta_i + \epsilon z$ ▷ `Modify parameters in place`
- **end**
- **return** $\boldsymbol{\theta}$

[Fine-Tuning Language Models with Just Forward Passes]

Zeroth-order Finetuning

北京大学 PEKING UNIVERSITY

Method	**Zero-shot / MeZO**	**ICL**	**Prefix FT**	**Full-parameter FT**
1.3B	1xA100 (4GB)	1xA100 (6GB)	1xA100 (19GB)	1xA100 (27GB)
2.7B	1xA100 (7GB)	1xA100 (8GB)	1xA100 (29GB)	1xA100 (55GB)
6.7B	1xA100 (14GB)	1xA100 (16GB)	1xA100 (46GB)	2xA100 (156GB)
13B	1xA100 (26GB)	1xA100 (29GB)	2xA100 (158GB)	4xA100 (316GB)
30B	1xA100 (58GB)	1xA100 (62GB)	4xA100 (315GB)	8xA100 (633GB)
66B	2xA100 (128GB)	2xA100 (134GB)	8xA100	16xA100

Zeroth-order Finetuning

	1.3B	2.7B	13B	30B	66B
MeZO (bsz=16)	0.815s (1)	1.400s (1)	2.702s (1)	5.896s (1)	12.438s (4)
MeZO (bsz=8)	0.450s (1)	0.788s (1)	1.927s (1)	4.267s (1)	7.580s (2)
FT (bsz=8)	0.784s (1)	1.326s (1)	13.638s (4)	45.608s (8)	84.098s (20)
	bspd=2, ga=4	bspd=2, ga=4	bspd=1, ga=2	bspd=1, ga=1	bspd=1, ga=1

Table 23: Wallclock time per step of different training methods. Numbers in brackets are numbers of GPUs required. It is measured on 80GB A100s with NVLink and InfiniteBand connections. The wallclock time is averaged over 100 steps. It is measured on the MultiRC task with the OPT family. We use a batch size ("bsz") of 8 for FT and 16 for MeZO (consistant with our main experiment setting). For comparison we also add MeZO with a batch size of 8. For FT (FSDP), we show the following additional information. "bspd": batch size per device. "ga": gradient accumulation steps. The effective batch size is bspd×ga× #GPUs. Note that for FT 66B, the effective batch size 20.

北京大学
PEKING UNIVERSITY

Thank you!

Kun Yuan homepage: https://kunyuan827.github.io/