北京大学
PEKING UNIVERSITY

Parameter-Efficient Fine-Tuning (PEFT)

Kun Yuan

Peking University

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

Finetune v.s. Prompt

北京大学 PEKING UNIVERSITY

Fine-tuning（微调）

优势:

- **任务适应性**：通过在特定任务的数据集上训练，模型可以更好地适应特定任务的需求。
- **性能提升**：在许多情况下，微调可以显著提高模型在特定任务上的性能。
- **端到端训练**：微调允许模型以端到端的方式学习任务，从而可能捕获更复杂的模式。

劣势:

- **数据需求**：需要大量标注数据来微调模型，这在某些情况下可能是限制因素。
- **过拟合风险**：在数据量有限的情况下，微调可能导致过拟合。
- **计算资源**：微调需要较多的计算资源，尤其是对于大型模型。

适用场景:

- **定制化任务**：当需要对模型进行特定定制，以适应特定的业务需求或领域时。
- **数据丰富**：当有足够的标注数据可用于训练时。
- **性能关键**：当任务性能至关重要，需要模型尽可能精确时。

Fine-tuning适用于有足够标注数据、需要高性能定制的场景

Finetune v.s. Prompt

北京大学 PEKING UNIVERSITY

Prompting（提示）

优势:

- **数据效率**：不需要或只需要很少的标注数据，适用于数据稀缺的场景。
- **泛化能力**：模型不需要针对特定任务进行调整，因此可能具有更好的泛化能力。
- **快速适应**：可以快速适应新任务，无需额外的训练过程。

劣势:

- **性能限制**：相比于微调，提示可能无法达到最佳性能，尤其是在复杂任务上。
- **提示工程**：需要精心设计提示，这可能需要领域知识和实验。
- **解释性挑战**：提示方法的输出可能不如微调模型那样直观易懂。

适用场景:

- **数据稀缺**：当标注数据有限或难以获取时。
- **快速迭代**：当需要快速适应新任务或领域时。
- **探索性任务**：当需要模型提供初步的、探索性的结果时。

Prompting适用于数据稀缺、需要快速适应新任务

Finetune v.s. RAG

北京大学 PEKING UNIVERSITY

Retrieval Augmented Generation (RAG)

优势:

- **上下文检索**：RAG结合了检索和生成的能力，可以从大量文本中检索相关信息来增强生成的内容。
- **知识更新**：RAG可以动态地检索最新的信息，使其在知识更新方面比仅依赖于预训练知识的模型更具优势。
- **减少偏见**：通过检索外部信息，RAG可以减少模型内部可能存在的偏见。

劣势:

- **检索质量**：生成内容的准确性高度依赖于检索系统的质量，如果检索到的信息不准确，生成的内容也可能不准确。
- **计算资源**：检索过程增加了额外的计算负担。
- **实时性**：对于需要实时响应的应用场景，检索过程可能会引入延迟。

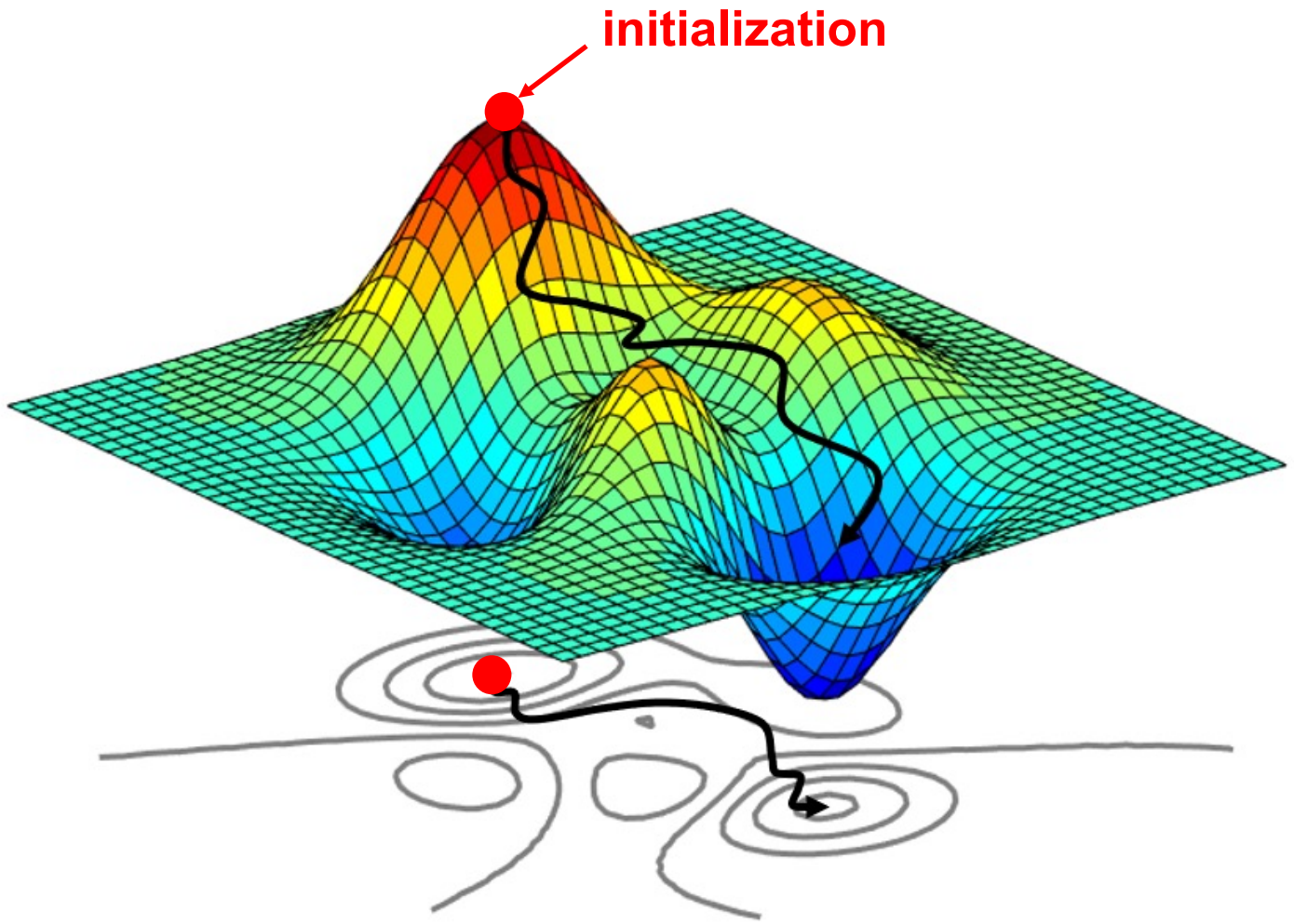
适用场景:

- **知识密集型任务**：当任务需要大量知识或事实信息时，如问答系统或文章生成。
- **实时信息**：当需要访问最新信息或实时数据时。
- **多样性需求**：当需要生成多样化的内容，而不只是依赖于模型内部的知识时。

RAG适用于需要动态检索外部知识、生成多样化内容的场景，且对实时性要求不高

Finetune is essentially retraining the model with nice initialization

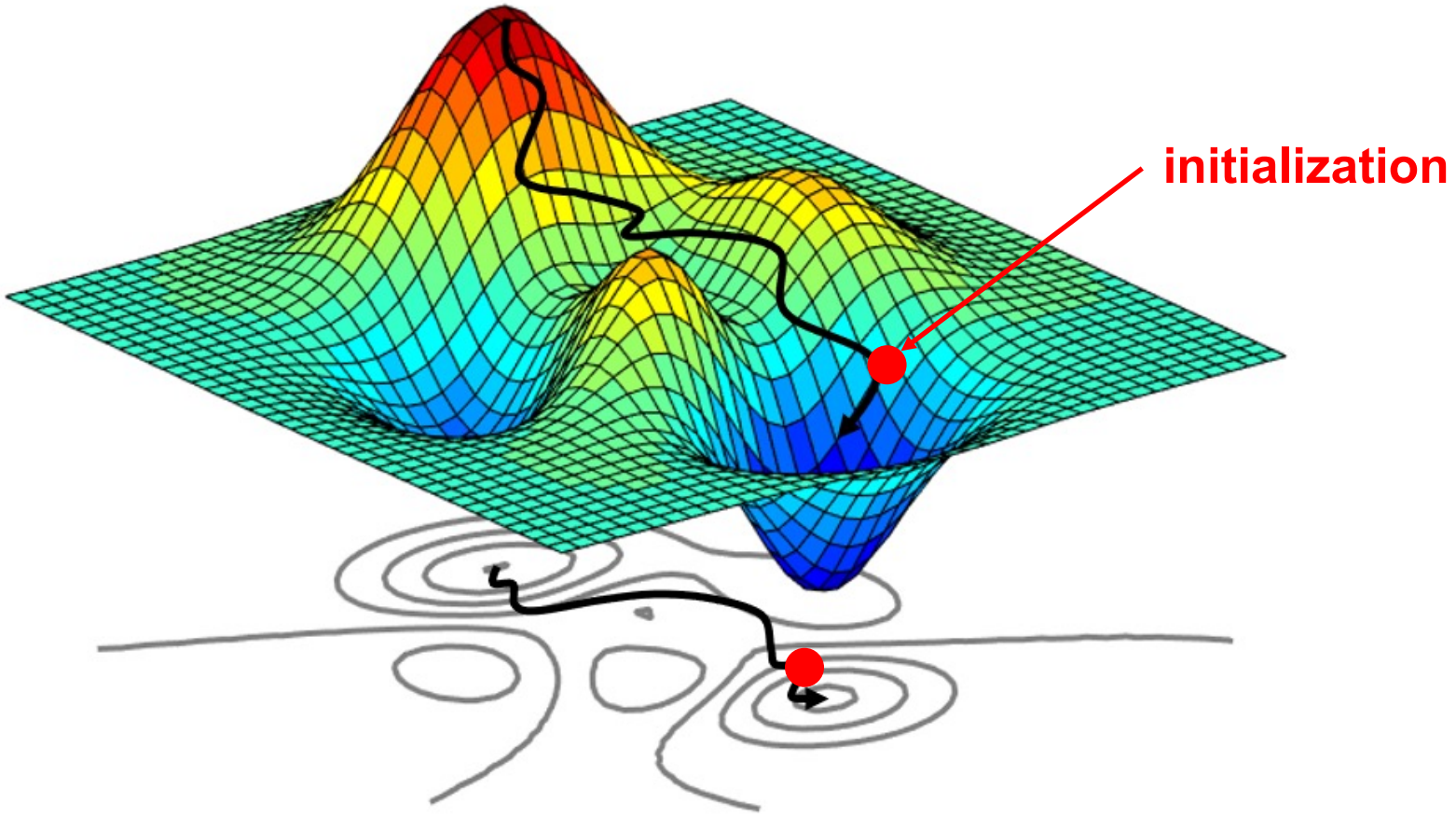
Pretrain



[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.

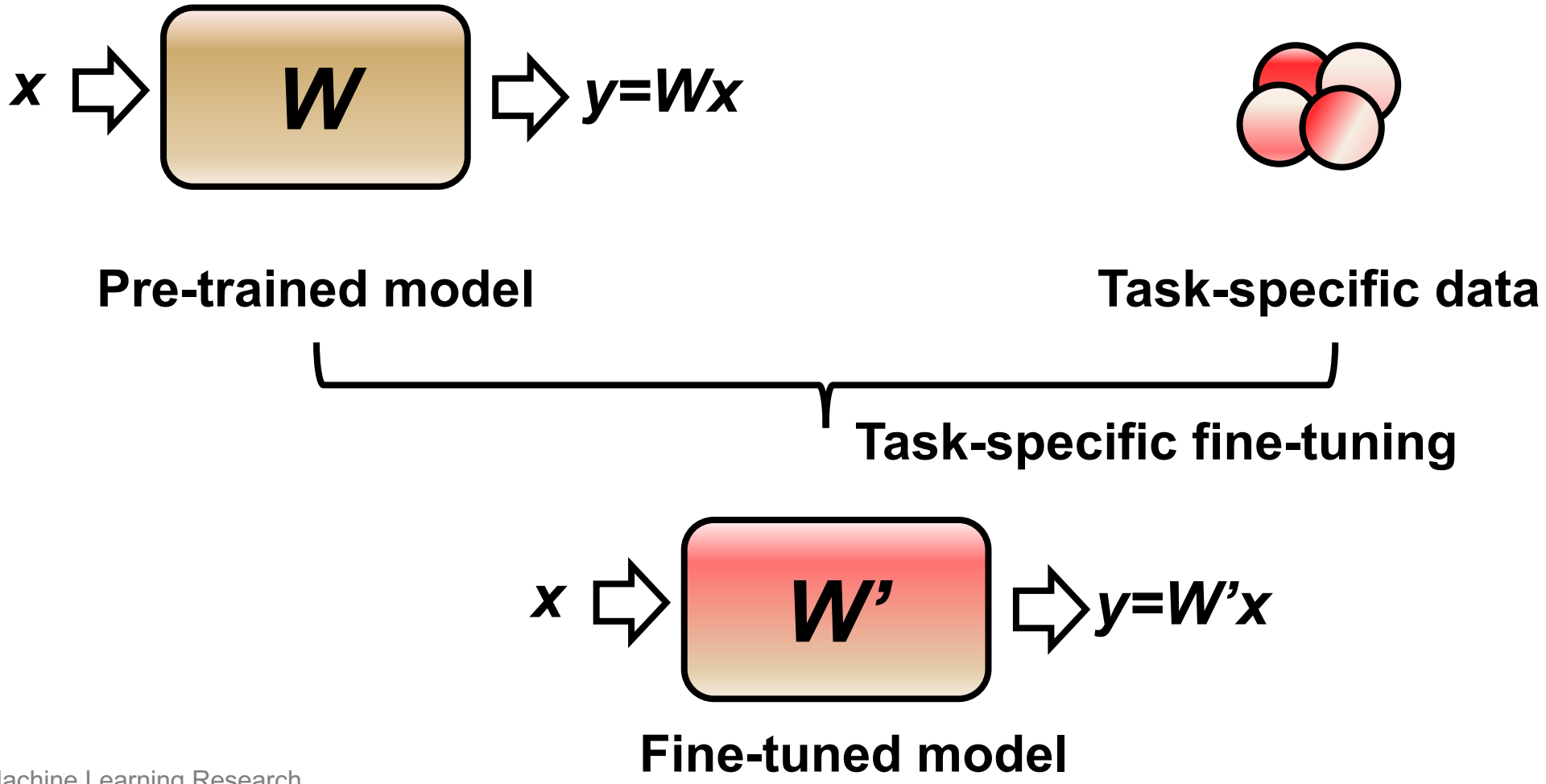
- **Data Requirements:** Finetuning an LLM effectively requires a significant amount of high-quality, domain-specific data. Gathering, cleaning, and preparing this data can be labor-intensive and costly.

- **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

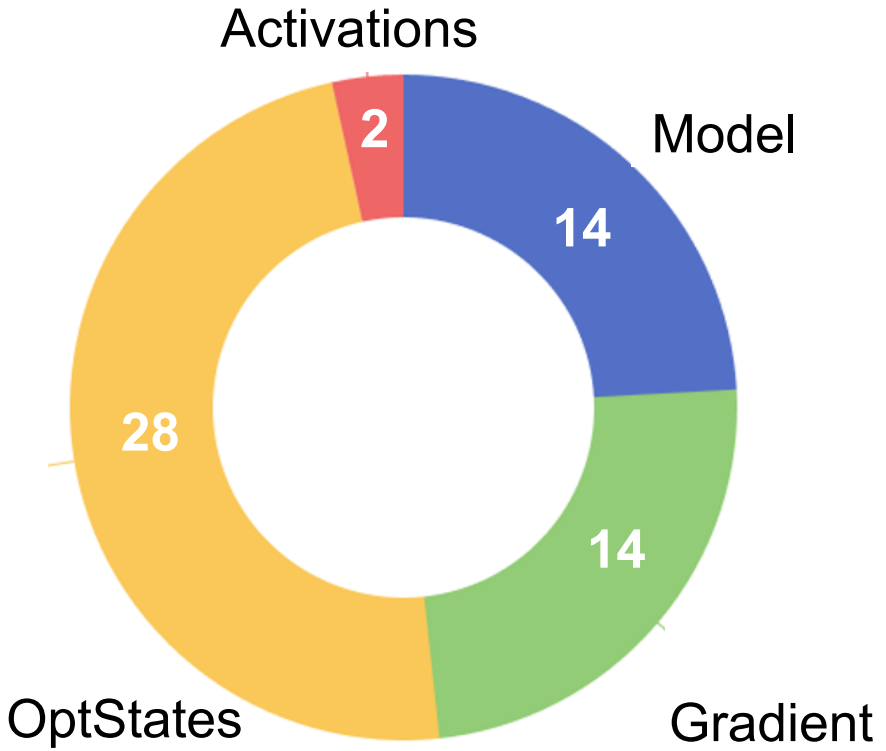
Model Fine-tuning

- Model fine-tuning helps to adapt the model to specific downstream tasks.



Why do we need parameter-efficient fine-tuning?

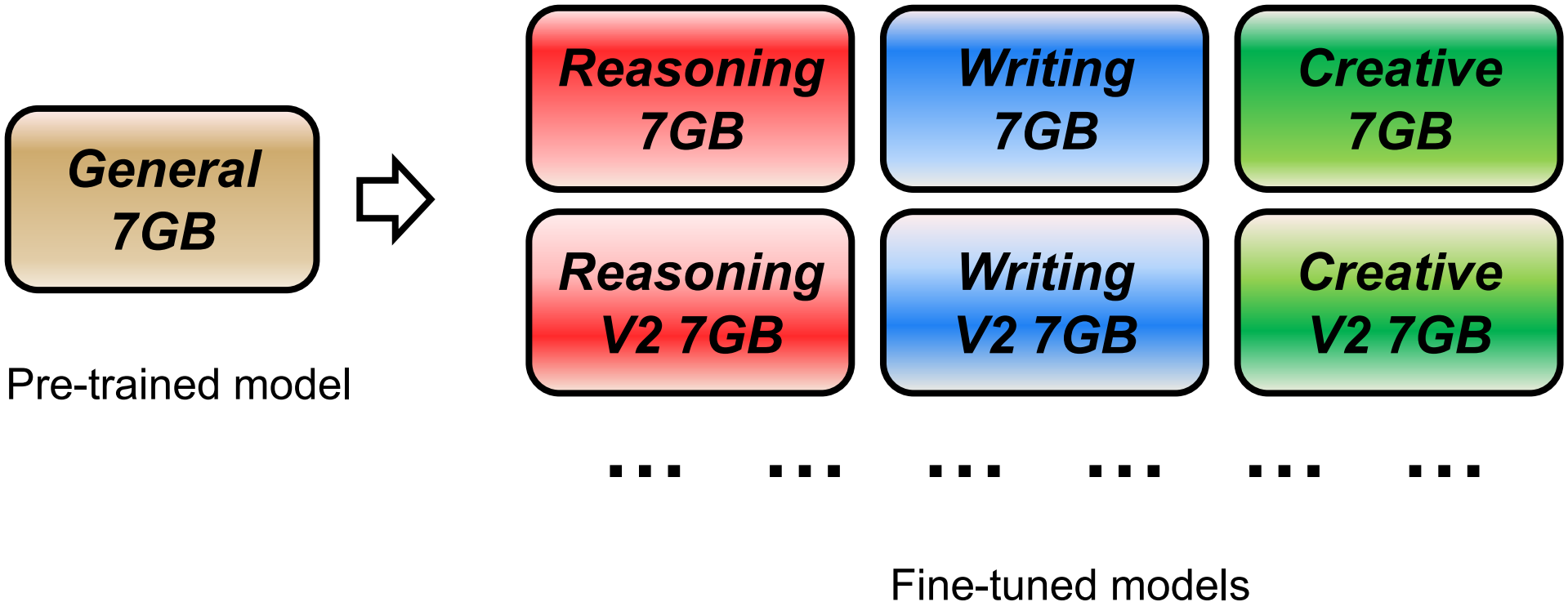
- Full fine-tuning requires **the same memory consumption as pre-training**.
  - Parameters: 7B
  - Model storage: 7B * 2 Bytes = 14 GB
  - Gradient storage: 14 GB
  - Optimizer states: 28 GB (using Adam)
  - Activation storage: 2 GB [Zhao et. al., 2024]
  - In total: **58 GB**



- Generally, models are pre-trained with lots of GPUs, which may not be easily accessible.

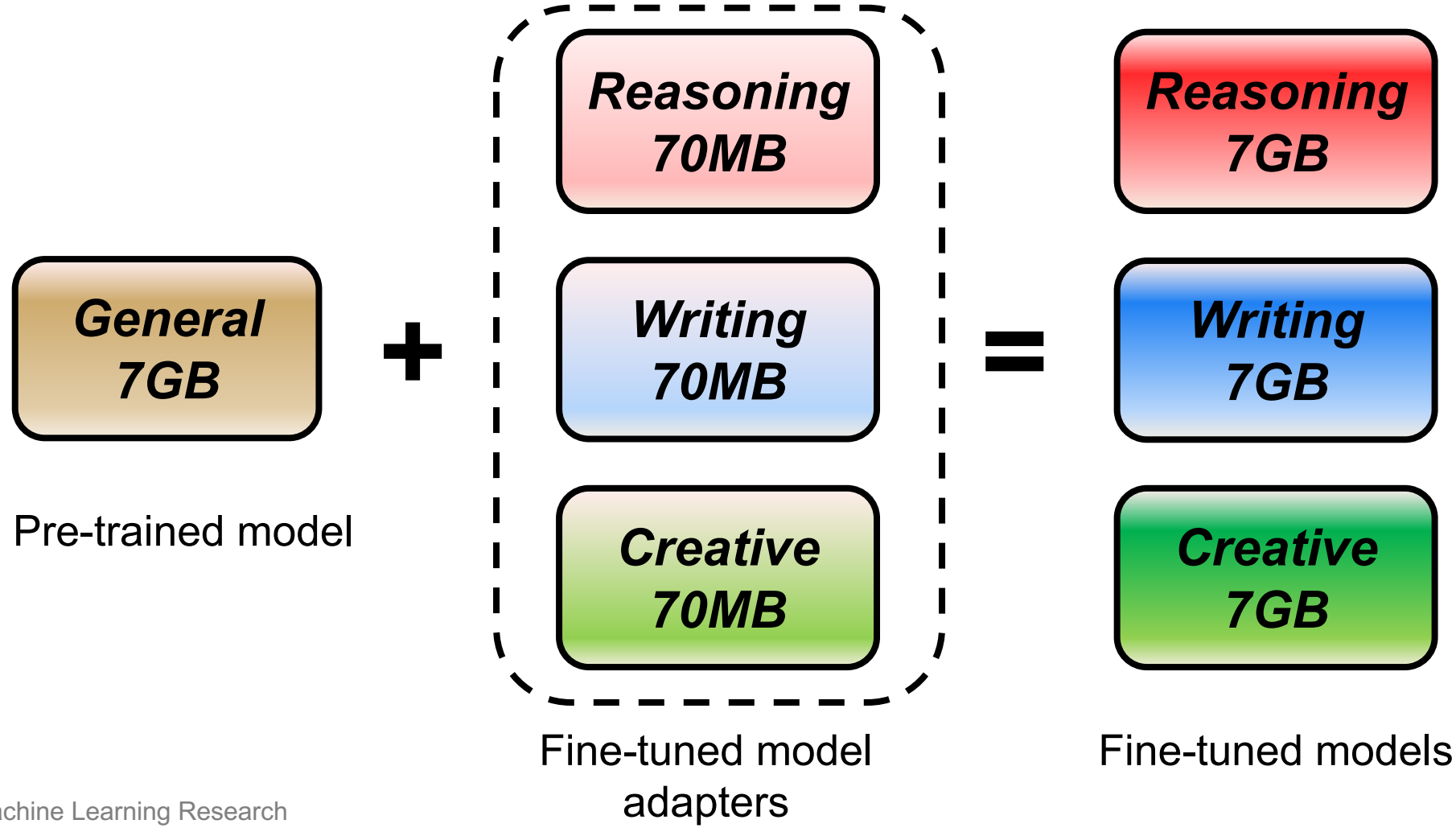
Why do we need parameter-efficient fine-tuning?

- For different downstream tasks / model versions, it is also **expensive** to store a full-size checkpoint for each fine-tuned model.



Parameter-efficient adapters

- Applying task-specific adapters, we can obtain task-specific models.



BitFit: Bias-terms Fine-tuning

北京大学 PEKING UNIVERSITY

- BitFit only fine-tunes the bias terms across all model layers.

Attention Projection:

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

Frozen **Trainable**

Multi-head Attention:

$$\mathbf{h}_1^{\ell} = att(\mathbf{Q}^{1,\ell}, \mathbf{K}^{1,\ell}, \mathbf{V}^{1,\ell}, .., \mathbf{Q}^{m,\ell}, \mathbf{K}^{m,\ell}, \mathbf{V}^{m,l})$$

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2021.

BitFit: Bias-terms Fine-tuning

北京大学 PEKING UNIVERSITY

- BitFit only fine-tunes the bias terms across all model layers.

Dropout Layer 1:

$$\mathbf{h}_2^\ell = \text{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right)$$

Layer Normalization 1:

$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell$$

GELU Activation:

$$\mathbf{h}_4^\ell = \text{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right)$$

Dropout Layer 2:

$$\mathbf{h}_5^\ell = \text{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right)$$

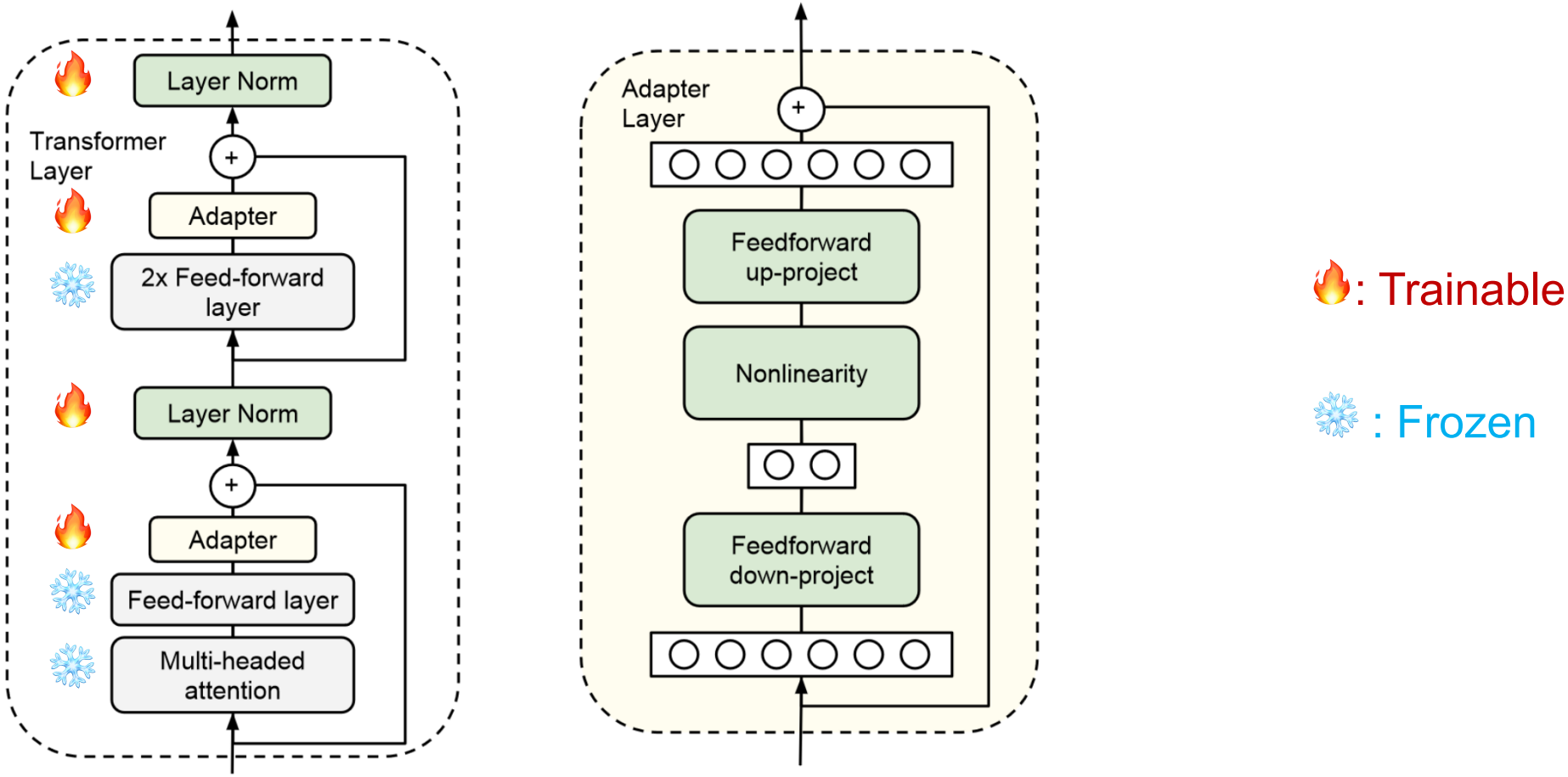
Layer Normalization 2:

$$\text{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell$$

Frozen **Trainable**

Adapter-H: Houlsby Architecture

- Adapter-H inserts additional layers in each transformer layer.



: Trainable

: Frozen

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. arXiv:1902.00751 [cs, stat], June 2019. URL http://arxiv.org/abs/1902. 00751.

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:

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

- Finetuning with low-rank adaptation:

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

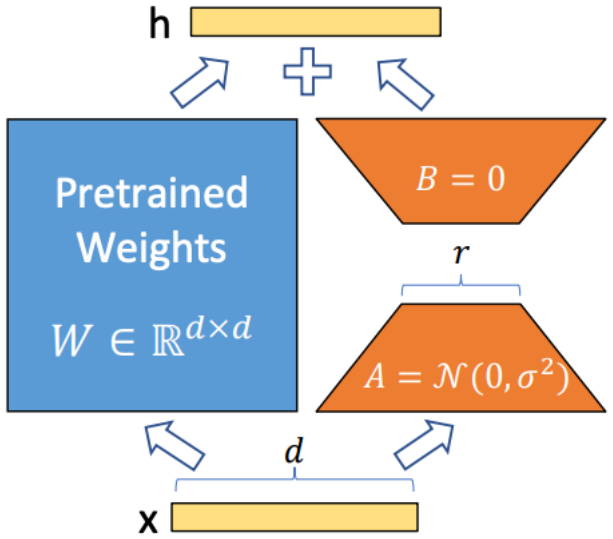
New knowledge and expertise are imposed through low rank matrices A and B

Finetuning LLM with low-rank adaptation

- Finetuning with low-rank adaptation:

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

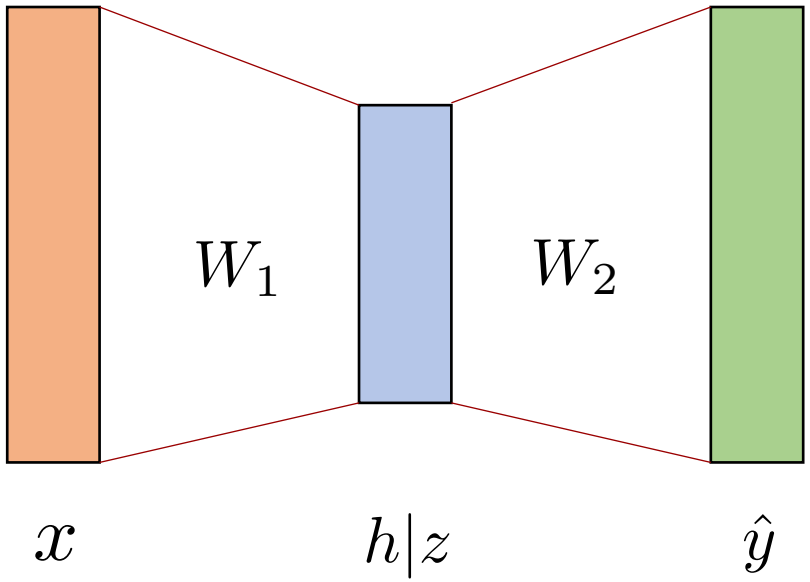
Only A and B are to be trained, while W is fixed



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

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

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

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

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

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

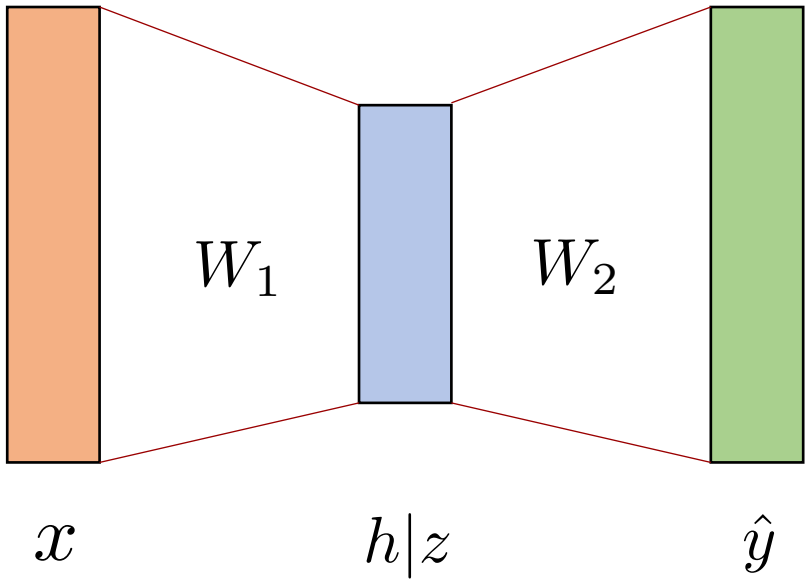
$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

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?

北京大学 PEKING UNIVERSITY

- Memory cost:

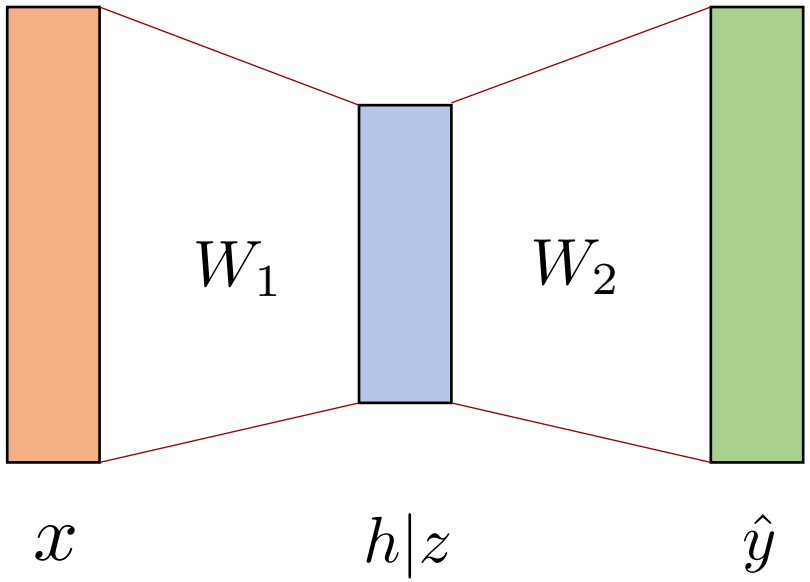
$$\text{Memory cost} = \text{Models} + \text{Grads} + \text{Optimizers} + \text{Activations}$$

- 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

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

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

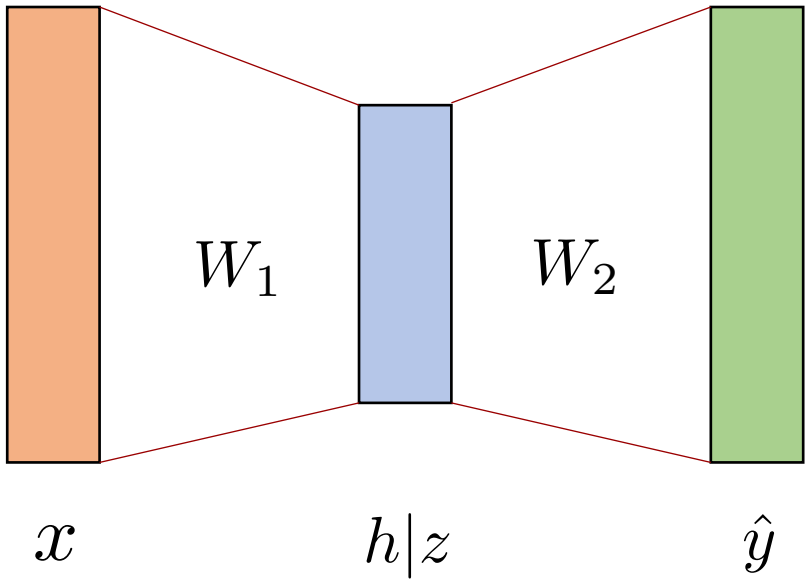
$$\frac{\partial f}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} z^T, \quad \frac{\partial f}{\partial z} = W_2^T \frac{\partial L}{\partial \hat{y}}$$

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

Backward

pd+2pq

LoRA: Computational cost



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

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

$$z = \sigma(h)$$

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

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

Forward

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

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

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

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

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

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

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

Does not save computations significantly!

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+

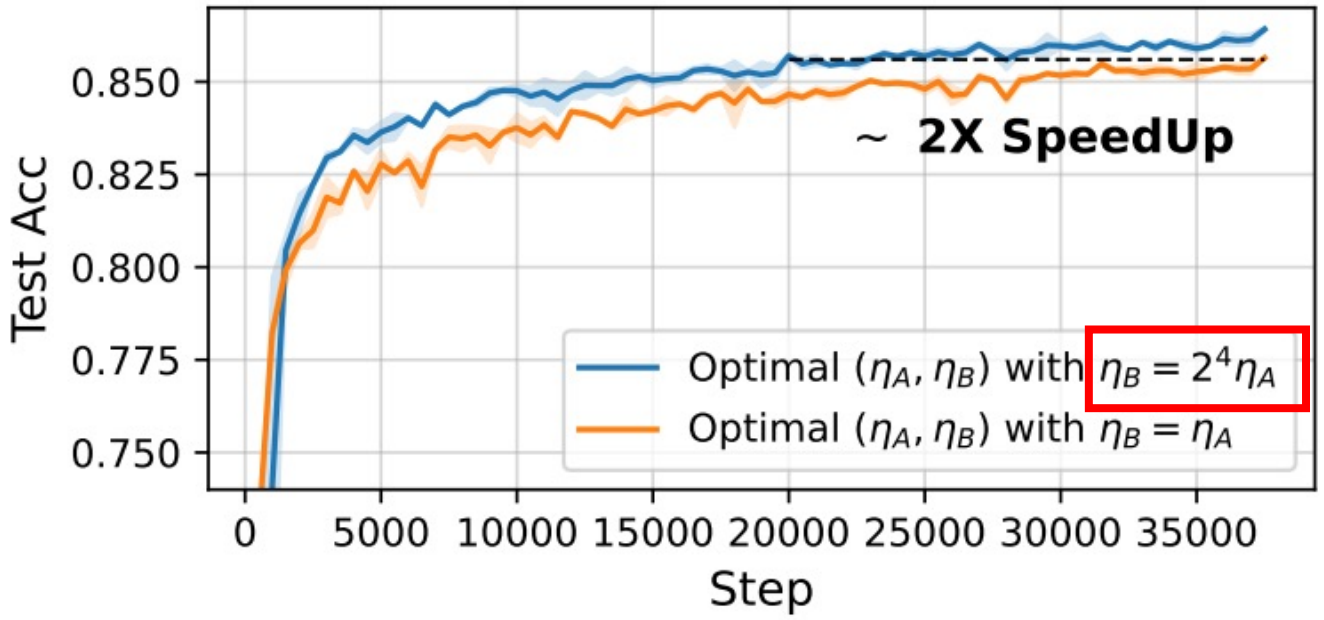
- Since one of the matrices A or B will be initialized to 0, it will be very slow for it to update
- We should use different learning rates for A and B; LoRA+

	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)

- A matrix can be decomposed into the magnitude and the direction

$$W = m\frac{V}{||V||_c} = ||W||_c\frac{W}{||W||_c}$$

- The magnitude and directional variations between W_0 and W_{FT} can be defined as follows:

$$\Delta M_{\mathrm{FT}}^t = \frac{\sum_{n=1}^{k} |m_{\mathrm{FT}}^{n,t} - m_0^n|}{k}$$

$$\Delta D_{\mathrm{FT}}^t = \frac{\sum_{n=1}^{k} (1 - \mathbf{cos}(V_{\mathrm{FT}}^{n,t}, W_0^n))}{k}$$

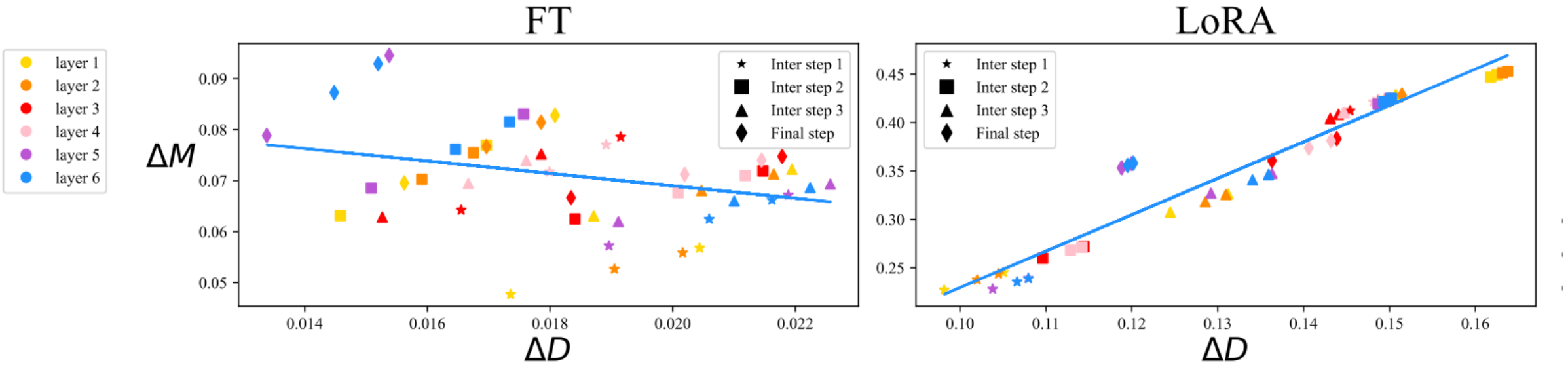
Weight-Decomposed Low-Rank Adaptation (DoRA)

- Similarly, the magnitude and directional variations between W_0 and W_{LoRA} are:

$$\Delta M^t_{\text{LoRA}} = \frac{\sum_{n=1}^{k} |m^{n,t}_{\text{LoRA}} - m^n_0|}{k}$$

$$\Delta D^t_{\text{LoRA}} = \frac{\sum_{n=1}^{k} (1 - \mathbf{cos}(V^{n,t}_{\text{LoRA}}, W^n_0))}{k}$$

Weight-Decomposed Low-Rank Adaptation (DoRA)



Slight-negatively proportional

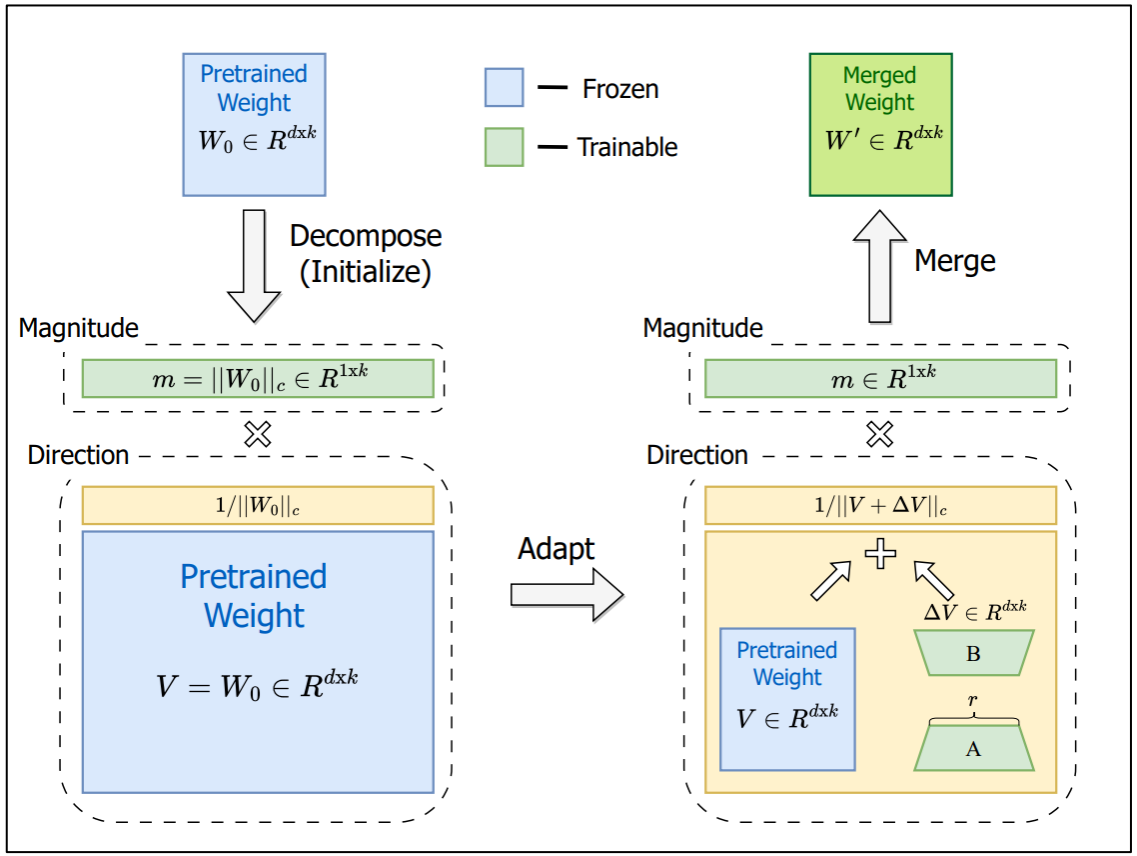
Positively proportional

- This reflects the difference between LoRA and FT

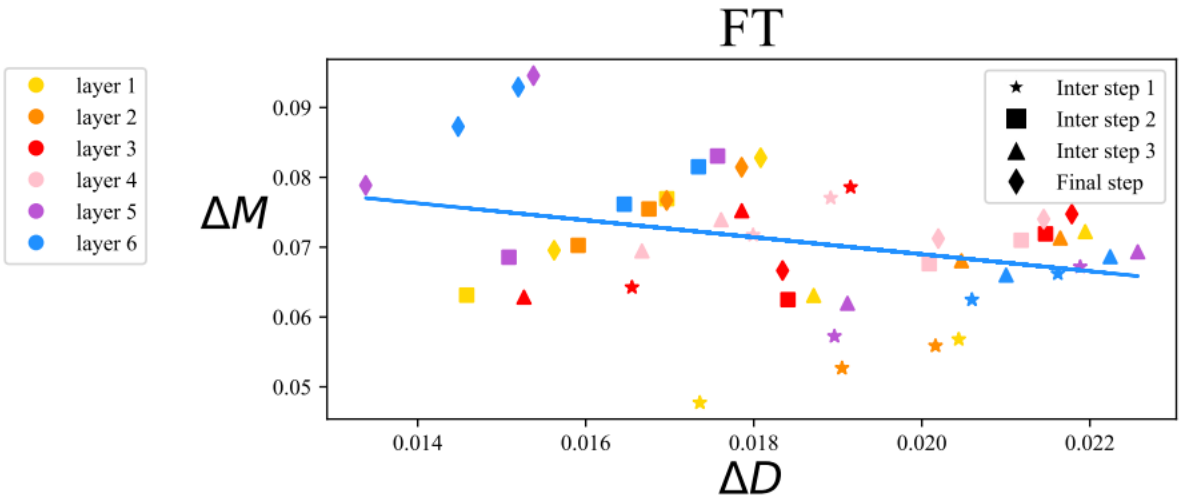
Weight-Decomposed Low-Rank Adaptation (DoRA)

- DoRA: train the magnitudes and directions separately

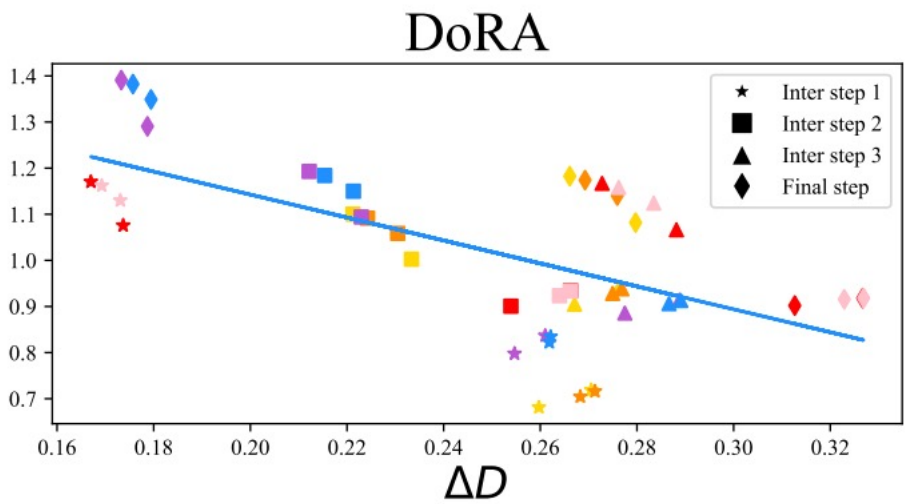
$$W' = \underline{m}\frac{V + \Delta V}{||V + \Delta V||_c} = \underline{m}\frac{W_0 + \underline{BA}}{||W_0 + \underline{BA}||_c}$$



Weight-Decomposed Low-Rank Adaptation (DoRA)



Slight-negatively proportional



Negatively proportional

- DoRA is consistent with FT

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**

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

北京大学 PEKING UNIVERSITY

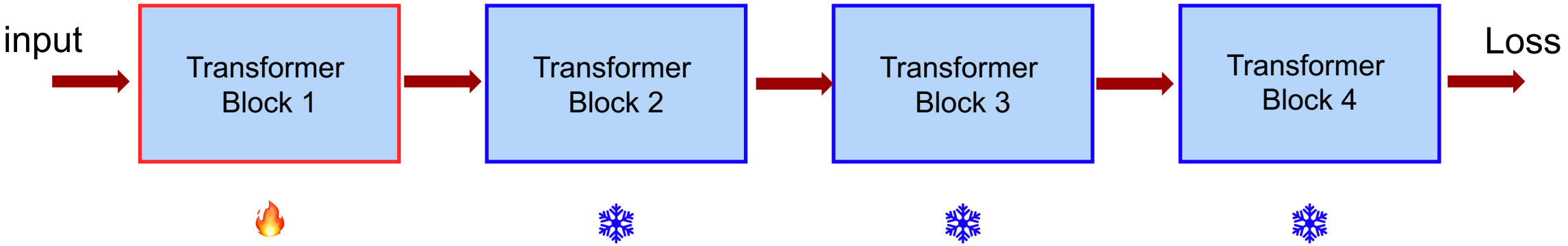
Algorithm 1 Layerwise Importance Sampling AdamW (**LISA**)

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**

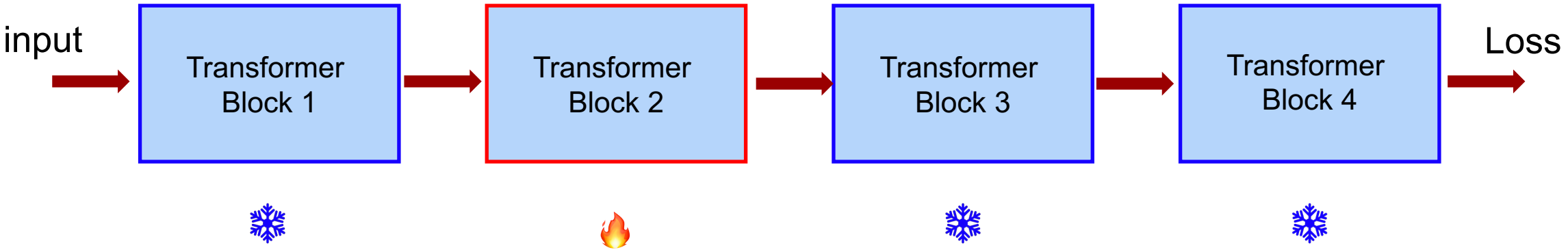
Block-wise training

Update the first block



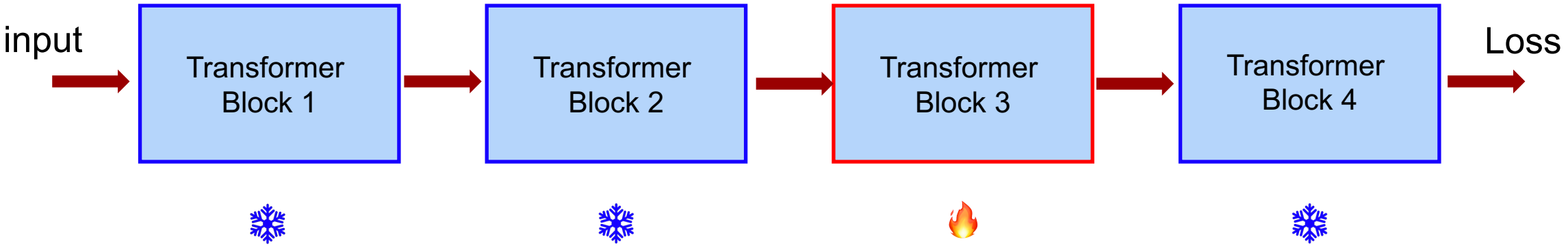
Block-wise training

Update the second block



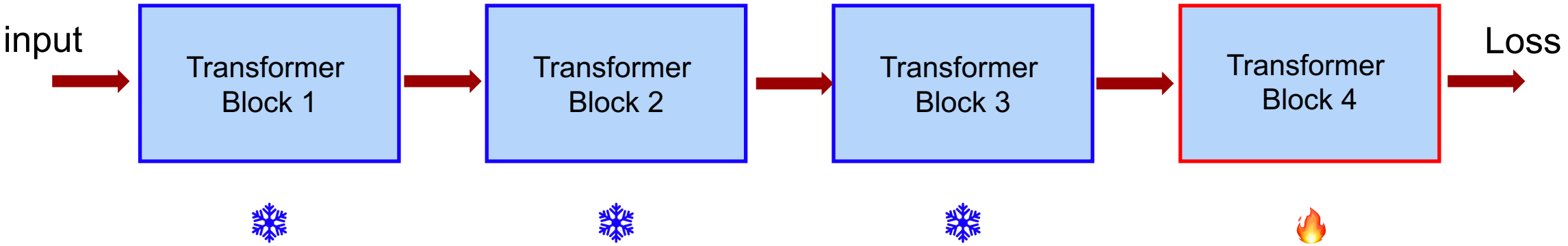
Block-wise training

Update the third block



Block-wise training

Update the forth block



Memory cost

北京大学 PEKING UNIVERSITY

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

Original

P $\boldsymbol{G}_t = \nabla F(\boldsymbol{X}_t; \boldsymbol{\xi}_t)$

2P
$$\boldsymbol{M}_t = (1-\beta_1)\boldsymbol{M}_{t-1} + \beta_1 \boldsymbol{G}_t$$
$$\boldsymbol{V}_t = (1-\beta_2)\boldsymbol{V}_{t-1} + \beta_2 \boldsymbol{G}_t \odot \boldsymbol{G}_t$$

P $\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \frac{\gamma}{\sqrt{\boldsymbol{V}_t} + \epsilon} \odot \boldsymbol{M}_t$

Block-wise

$G_{t,i} = \nabla_i F(\boldsymbol{X}_t; \xi_t)$ **P/L**

2P/L
$$M_{t,i} = (1-\beta_1)M_{t-1,i} + \beta_1 G_{t,i}$$
$$V_{t,i} = (1-\beta_2)V_{t-1,i} + \beta_2 G_{t,i} \odot G_{t,i}$$

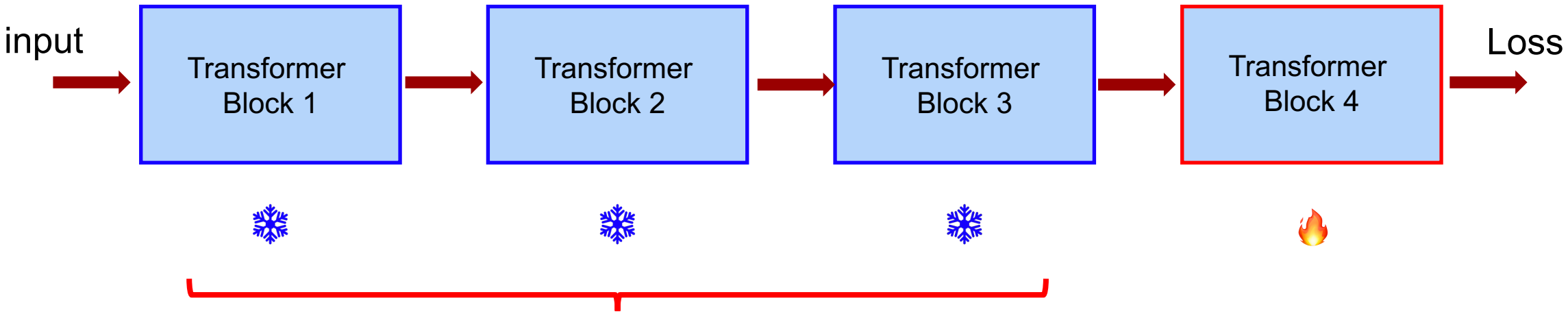
$X_{t+1,i} = X_{t,i} - \frac{\gamma}{\sqrt{V_{t,i}} + \epsilon} \odot M_{t,i}$ **P**

Block-wise training can save gradients and optimization states significantly

Y. Chen, et. al., *Enhancing Zeroth-Order Fine-tuning for Language Models with Low-Rank Structures*, 2024

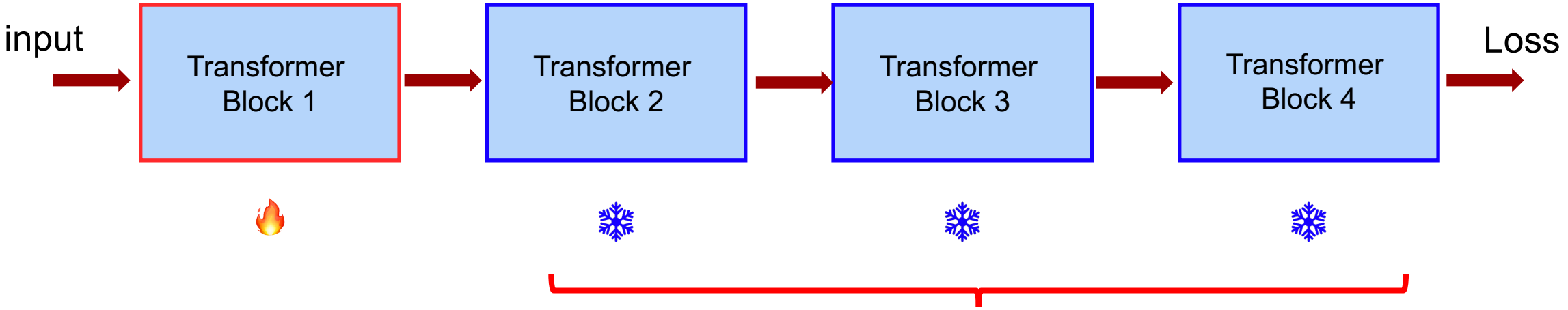
Memory cost

- Block-wise training can also save activations, but depends on which block has been activated



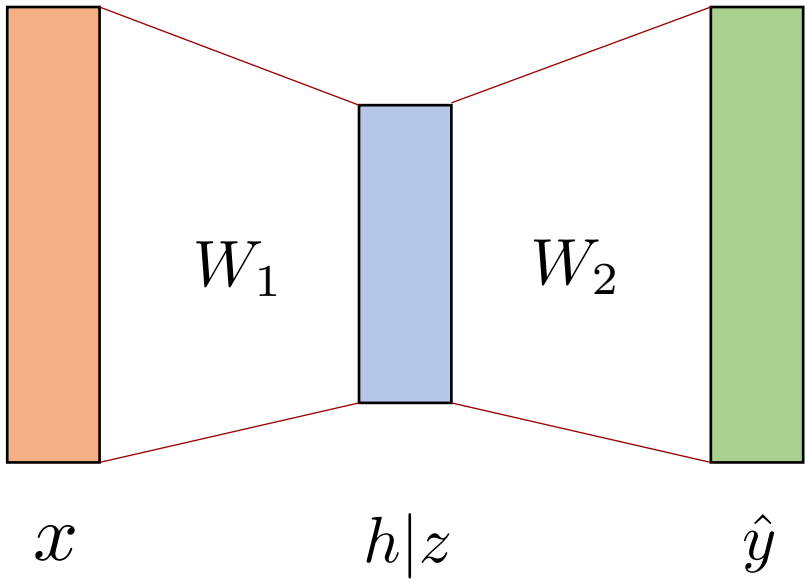
Memory cost

- Block-wise training can also save activations, but depends on which block has been activated



Most activations cannot be saved because we need to compute the gradient of the activations

Example: 2-layer MLP



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

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

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

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

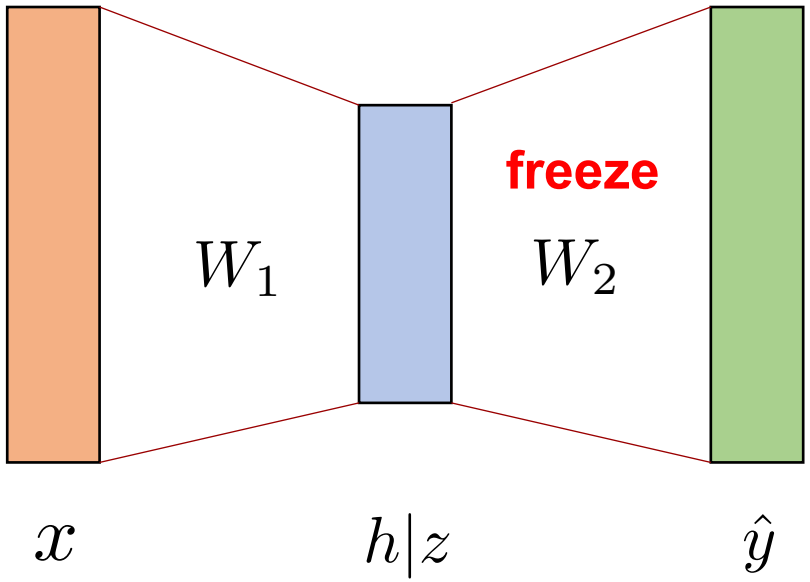
$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

Backward

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

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

Example: 2-layer MLP



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 $\hat{y}$

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

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

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

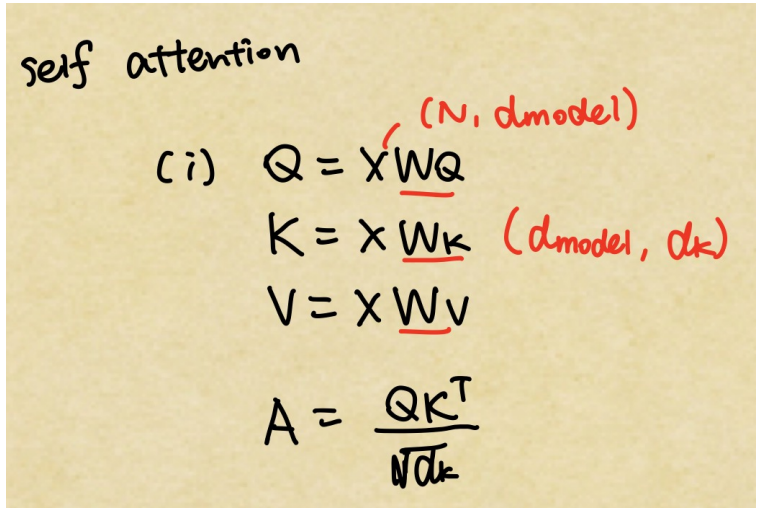
$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

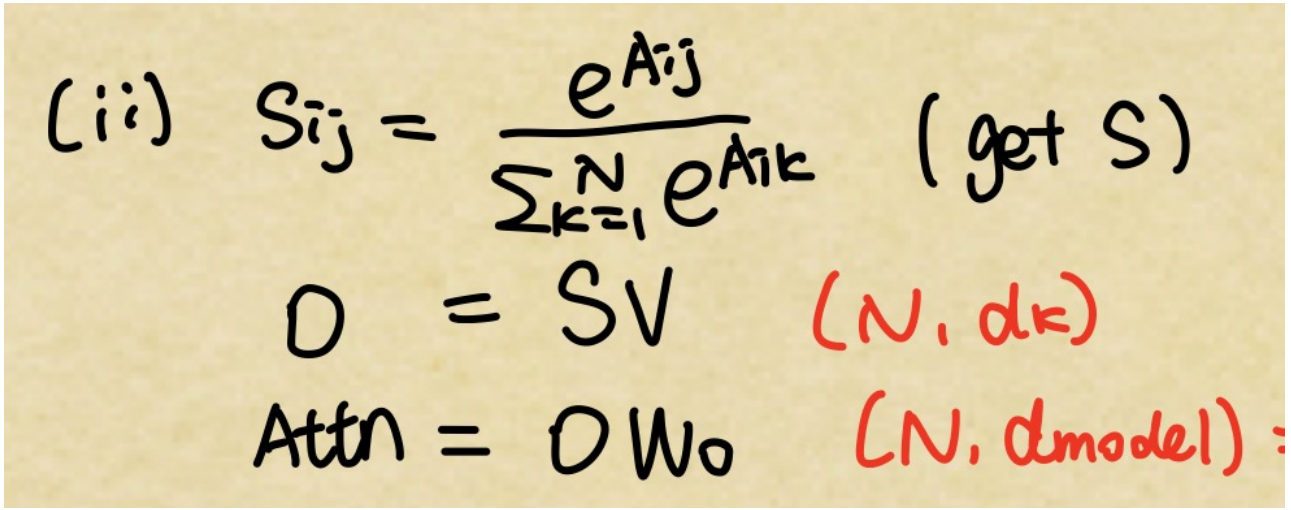
Backward

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

Activation h is still needed

Example: Attention





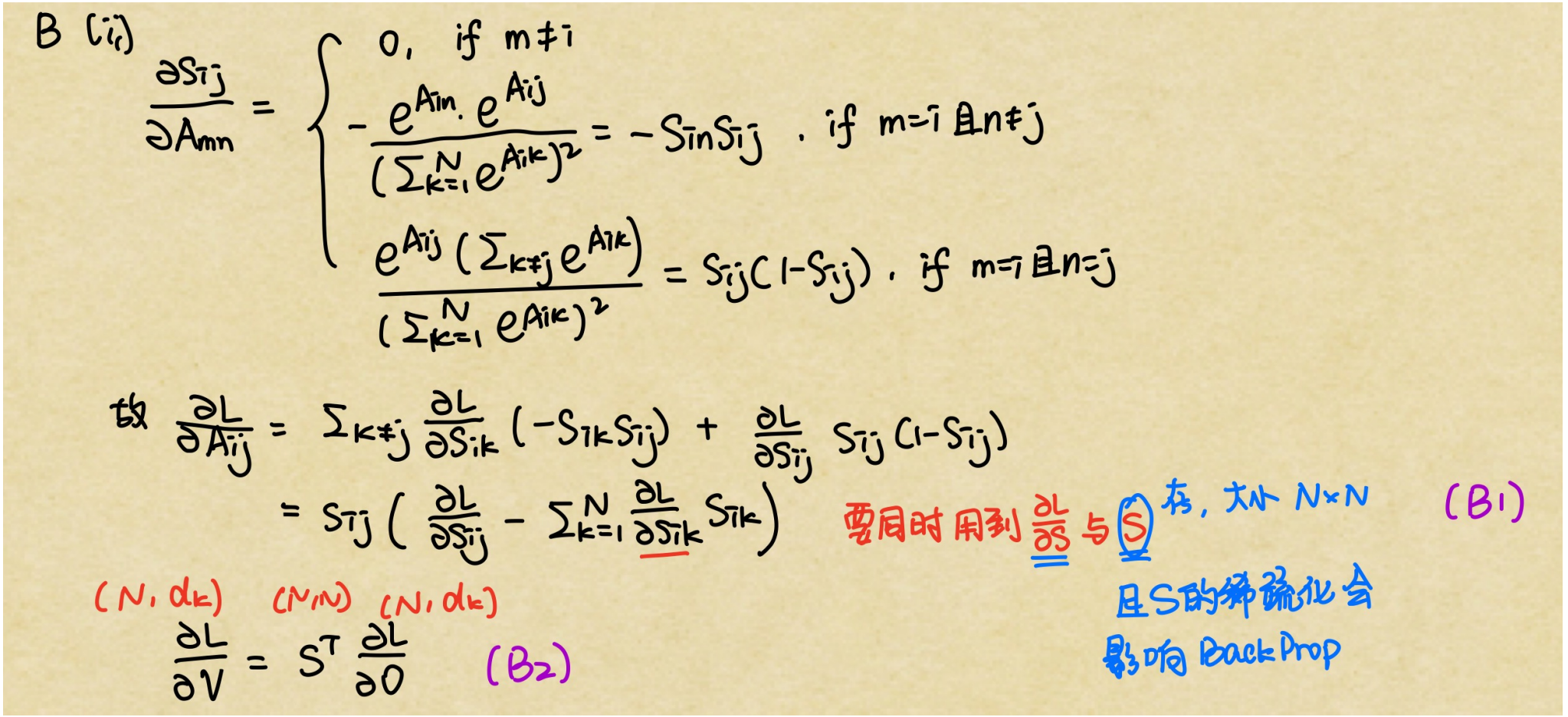
Store Q, K, V

Do not Store X

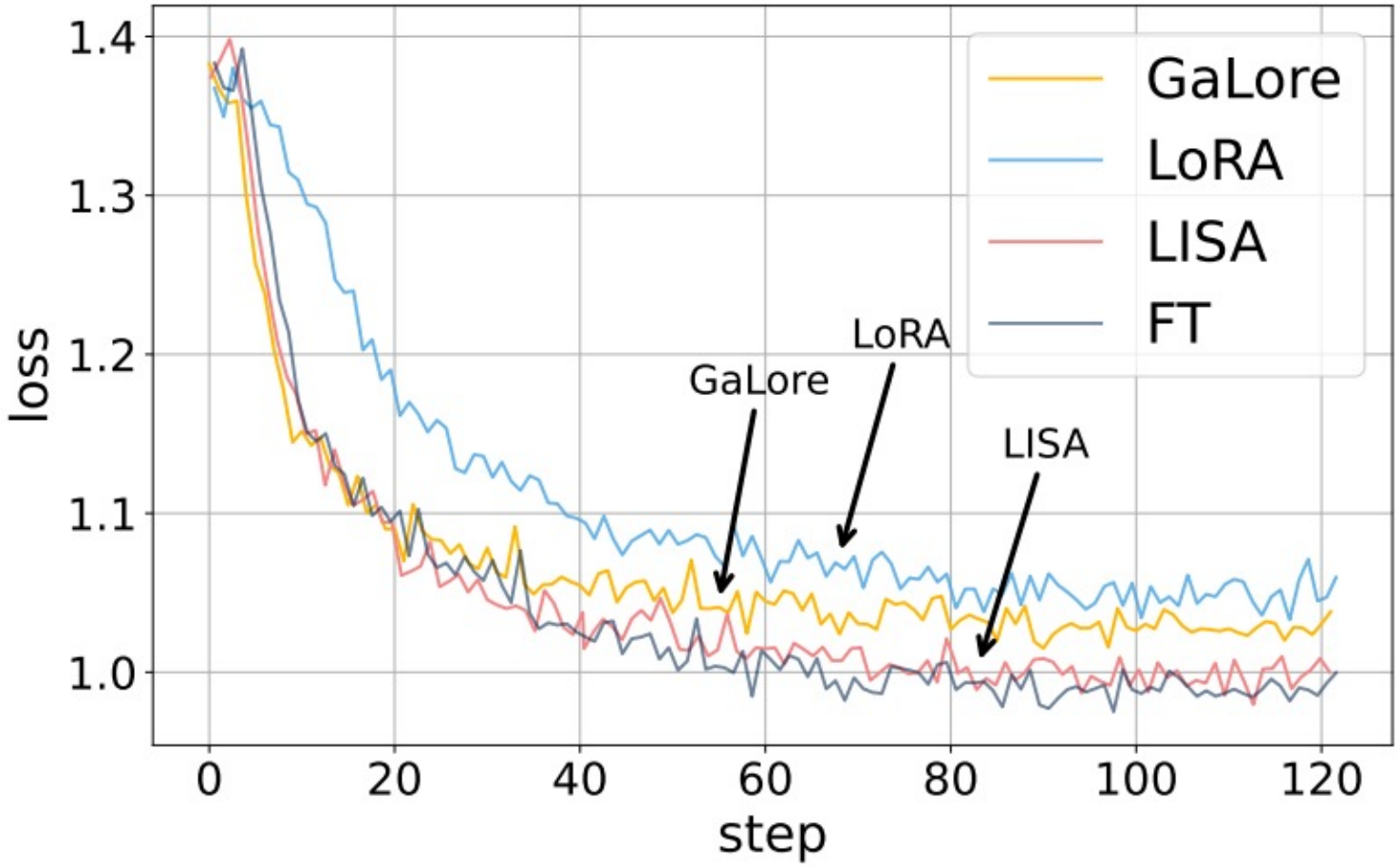
Store S, V

Do not store A, O

Example: Attention



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