北京大学
PEKING UNIVERSITY

Block-wise Training in LLMs

Kun Yuan (袁 坤)

Center for Machine Learning Research @ Peking University

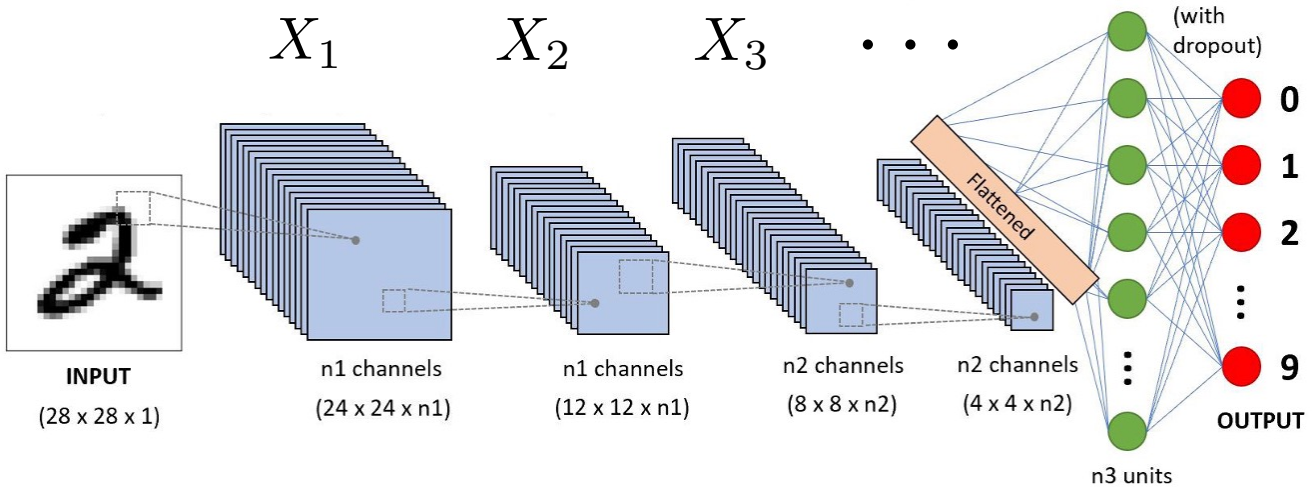
北京大学
PEKING UNIVERSITY

PART 01

Basics and Motivation

LLM pretraining is essentially solving stochastic optimization

- The model weights in neural networks are a set of matrices $\boldsymbol{X} = \{X_\ell\}_{\ell=1}^{L}$



- Let $h(\boldsymbol{X};\xi)$ be the language model; $\hat{y} = h(\boldsymbol{X};\xi)$ is the predicted token

cross entropy

LLM cost function: $$\boldsymbol{X}^\star = \arg\min_{\boldsymbol{X}} \left\{ \mathbb{E}_{\xi\sim\mathcal{D}} \left[L(h(\boldsymbol{X};\xi), y) \right] \right\}$$

data distribution pred. token real token

LLM pretraining is essentially solving stochastic optimization

- If we define $\boldsymbol{\xi} = (\xi, y)$ and $F(\boldsymbol{X}; \boldsymbol{\xi}) = L(h(\boldsymbol{X}; \xi), y)$, the LLM problem becomes

$$\text{Stochastic optimization:} \quad \boldsymbol{X}^\star = \arg\min_{\boldsymbol{X}} \left\{ \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{D}} \left[F(\boldsymbol{X}; \boldsymbol{\xi}) \right] \right\}$$

- In other words, LLM pretraining is essentially solving a stochastic optimization problem

- Adam is the standard approach in LLM pretraining

Optimizer states

$$\boldsymbol{G}_t = \nabla F(\boldsymbol{X}_t; \boldsymbol{\xi}_t) \qquad \text{(stochastic gradient)}$$

$$\boldsymbol{M}_t = (1-\beta_1)\boldsymbol{M}_{t-1} + \beta_1 \boldsymbol{G}_t \qquad \text{(first-order momentum)}$$

$$\boldsymbol{V}_t = (1-\beta_2)\boldsymbol{V}_{t-1} + \beta_2 \boldsymbol{G}_t \odot \boldsymbol{G}_t \qquad \text{(second-order momentum)}$$

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \frac{\gamma}{\sqrt{\boldsymbol{V}_t} + \epsilon} \odot \boldsymbol{M}_t \qquad \text{(adaptive SGD)}$$

Memory cost to pre-train LLMs

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

- Given a **model** with **P** parameters, **gradient** will consume **P** parameters, and **optimizer states** will consume **2P** parameters; **4P parameters in total.**

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

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

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

Optimizer states introduces significant memory cost

Memory cost to pre-train LLMs

北京大学 PEKING UNIVERSITY

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

- Activations are auxiliary variables to facilitate the gradient calculations

Consider a linear neural network

$$z_i = X_i z_{i-1}, \forall\, i = 1, \cdots, L$$
$$f = \mathcal{L}(z_i; y)$$

The gradient is derived as follows

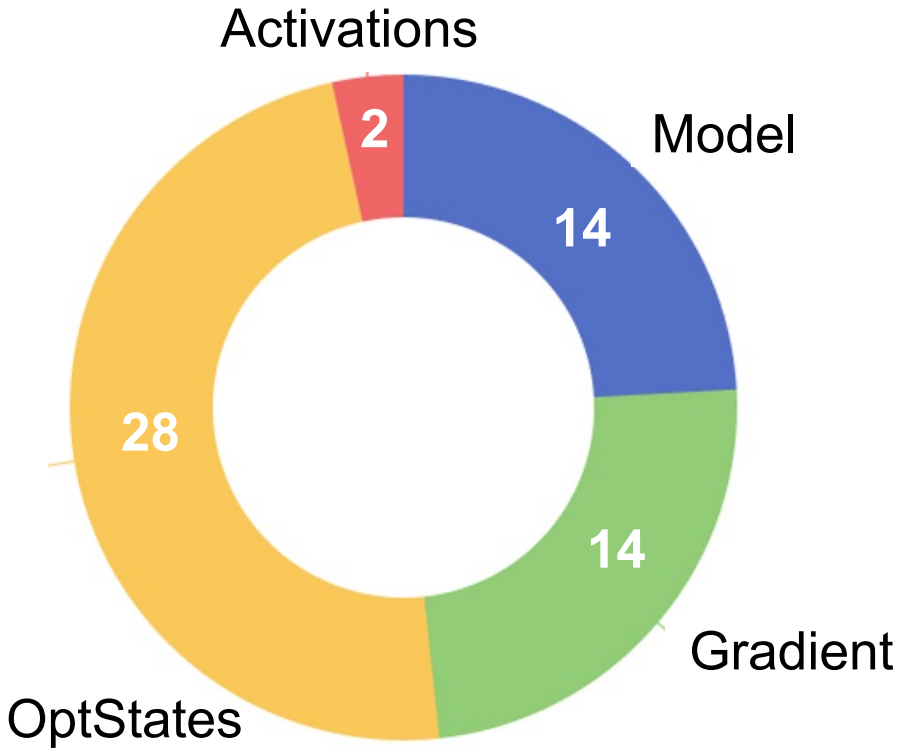
$$\frac{\partial f}{\partial X_i} = \frac{\partial f}{\partial z_i} z_{i-1}^\top$$

Need to store activations $z_1, z_2, \cdots, z_L$

- The size of activations depends on sequence length and batch size

Minimum memory requirement

- Pretrain LLaMA 7B model (BF16) from scratch with **a single batch size** requires
  - 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**



Zhao et. al., *GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection*, ICML 2024

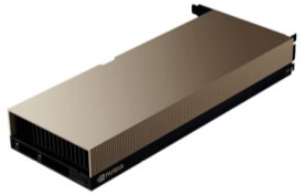
Minimum memory requirement: LLaMA 7B

北京大学 PEKING UNIVERSITY

- Pretrain LLaMA 7B model (BF16) from scratch with **a single batch size** requires

  - 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**

RTX 4090: 24GB



A100 80G

The minimum requirement is A100 * 1

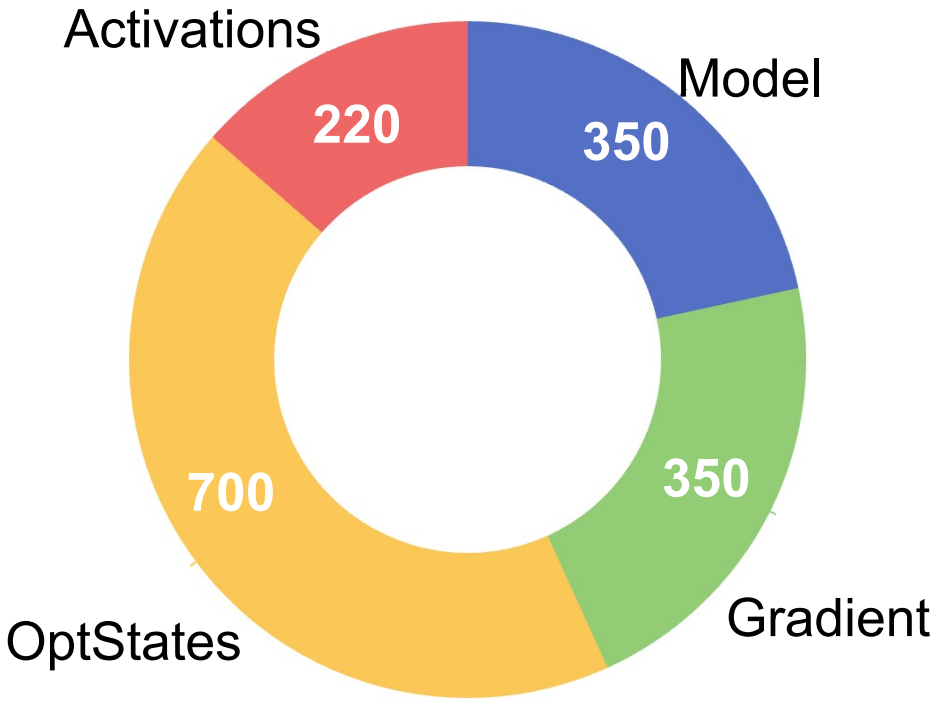
This slide is from Jiawei Zhao's talk on "Memory-efficient LLM Training"

Minimum memory requirement: GPT-3

北京大学 PEKING UNIVERSITY

- Pretrain GPT-3 model (BF16) from scratch with **a single batch size** requires

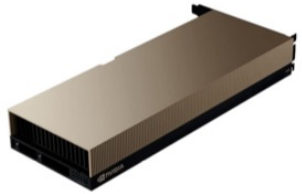
  - Parameters: 175B
  - Model storage: 175B * 2 Bytes = 350 GB
  - Gradient storage: 350 GB
  - Optimizer states: 700 GB (using Adam)
  - Activation storage: ~220 GB
  - In total: **1620 GB**



Minimum memory requirement: GPT-3

北京大学 PEKING UNIVERSITY

- Pretrain GPT-3 model (BF16) from scratch with **a single batch size** requires
  - Parameters: 175B
  - Model storage: 175B * 2 Bytes = 350 GB
  - Gradient storage: 350 GB
  - Optimizer states: 700 GB (using Adam)
  - Activation storage: ~220 GB
  - In total: **1620 GB**



A100 80G

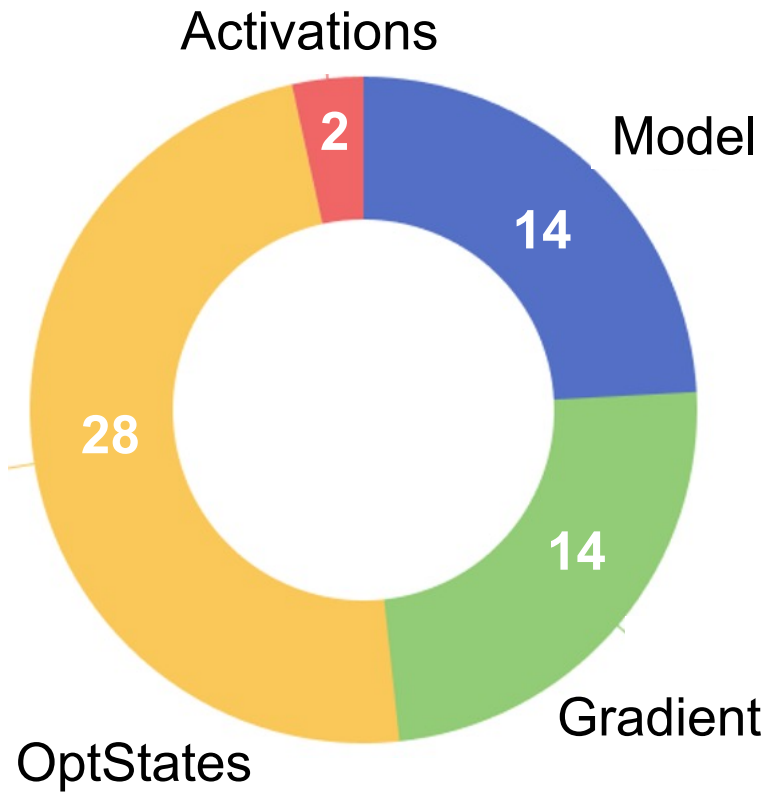
x 21

The minimum requirement is A100 * 21

Very expensive!

Memory-efficient algorithm is in urgent need

- With memory-efficient algorithms, we can
  - **Train larger models** on limited computing resources
  - Use a larger training batch size to **improve throughput**
- Activation-incurred memory is **relatively minor** when using a single batch size
- Gradient-incurred memory can be **removed** by layer-wise calculation and dropping
- How to save memory caused by optimizer states?



PART 02

Block-wise training

Block-wise training

- 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

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

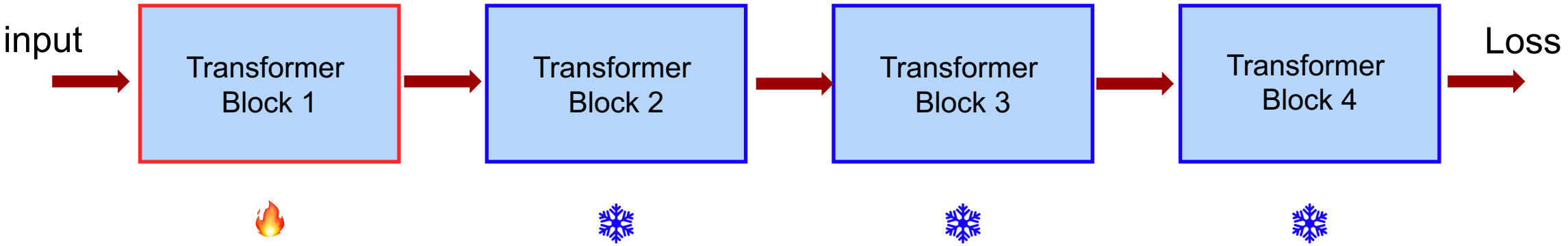
Require: number of layers N_L, number of iterations T, sampling period K, number of sampled layers γ, initial learning rate η_0

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**

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

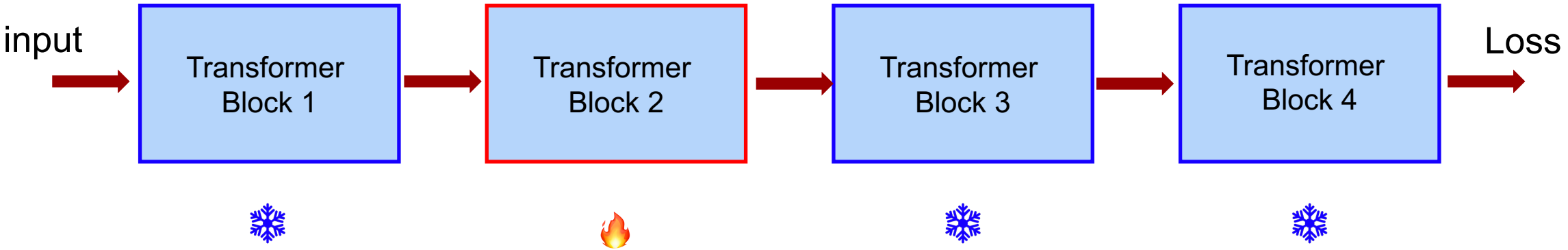
Block-wise training

Update the first block



Block-wise training

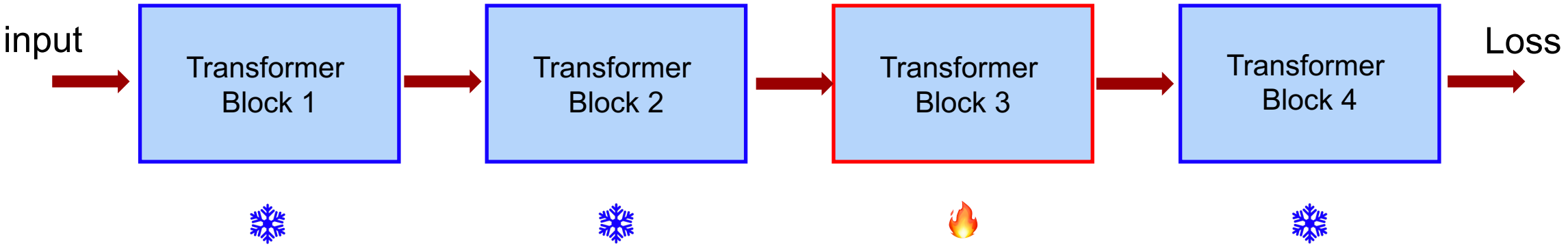
Update the second block



Block-wise training

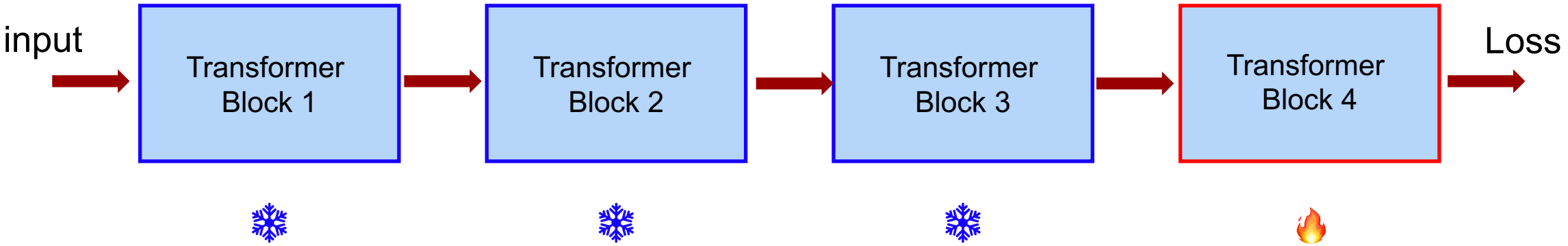
北京大学 PEKING UNIVERSITY

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

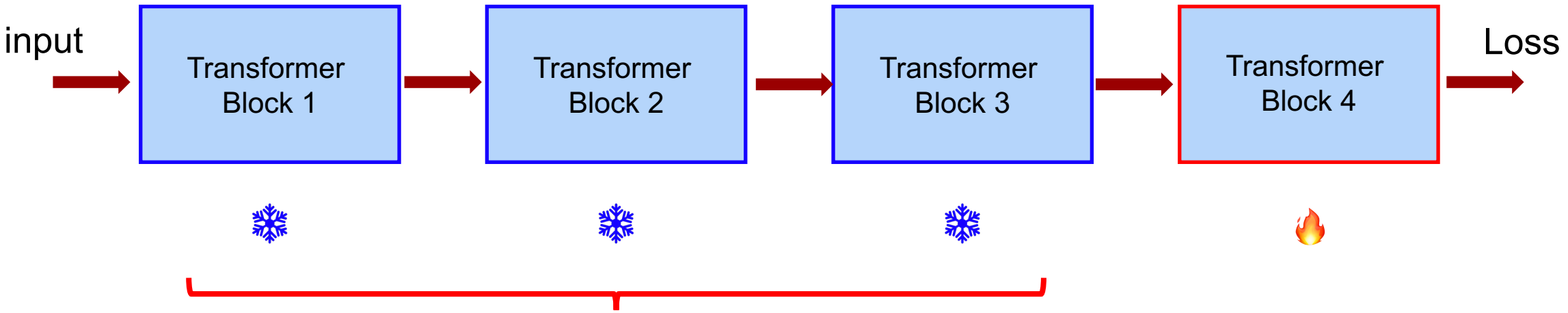
$X_{t+1,i} = X_{t,i} - \frac{\gamma}{\sqrt{V_{t,i}} + \epsilon} \odot M_{t,i}$ **P**

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

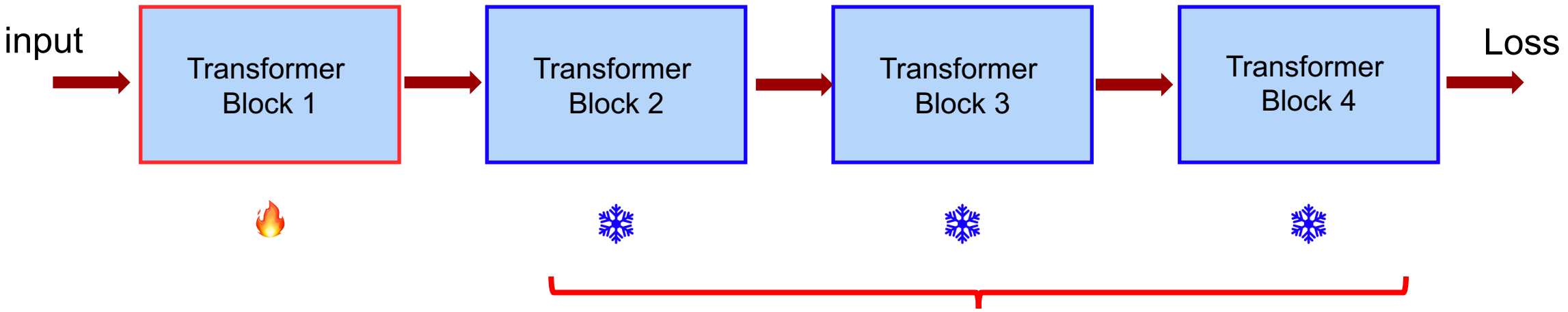
Memory cost

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



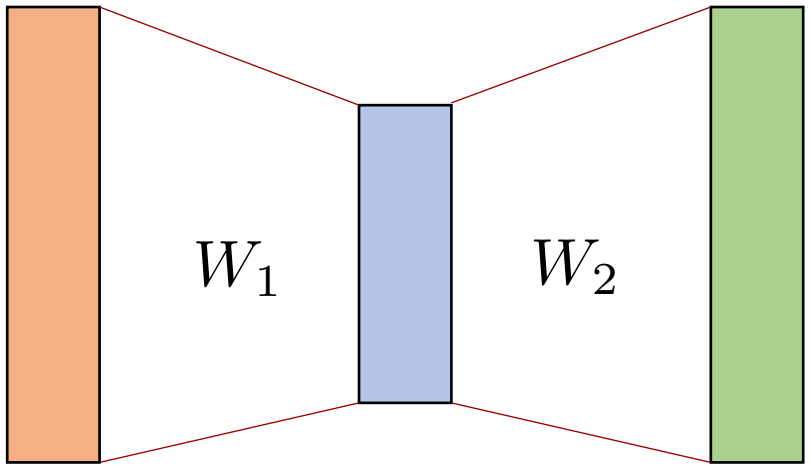
Memory cost

- 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

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

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

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

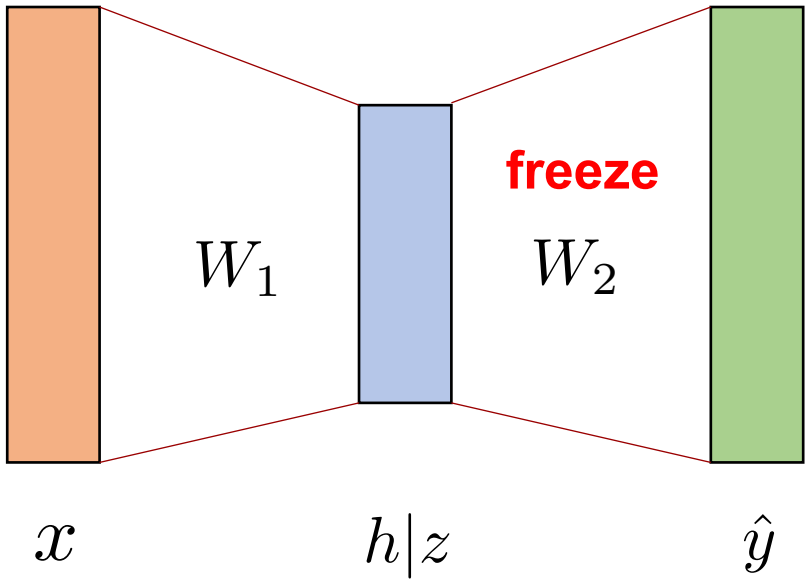
$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

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 $\hat{y}$

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

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

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

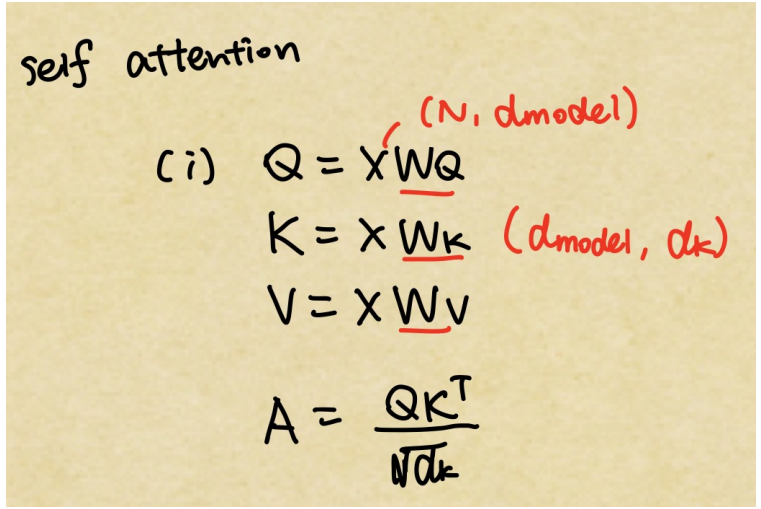
$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

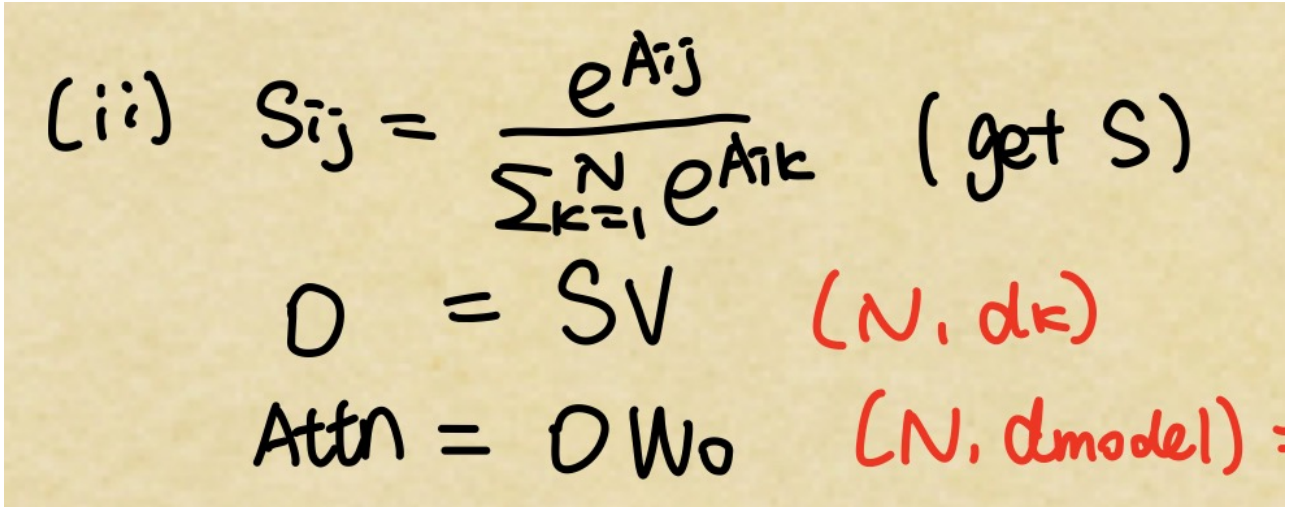
Backward

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

Activation h is still needed

Example: Attention





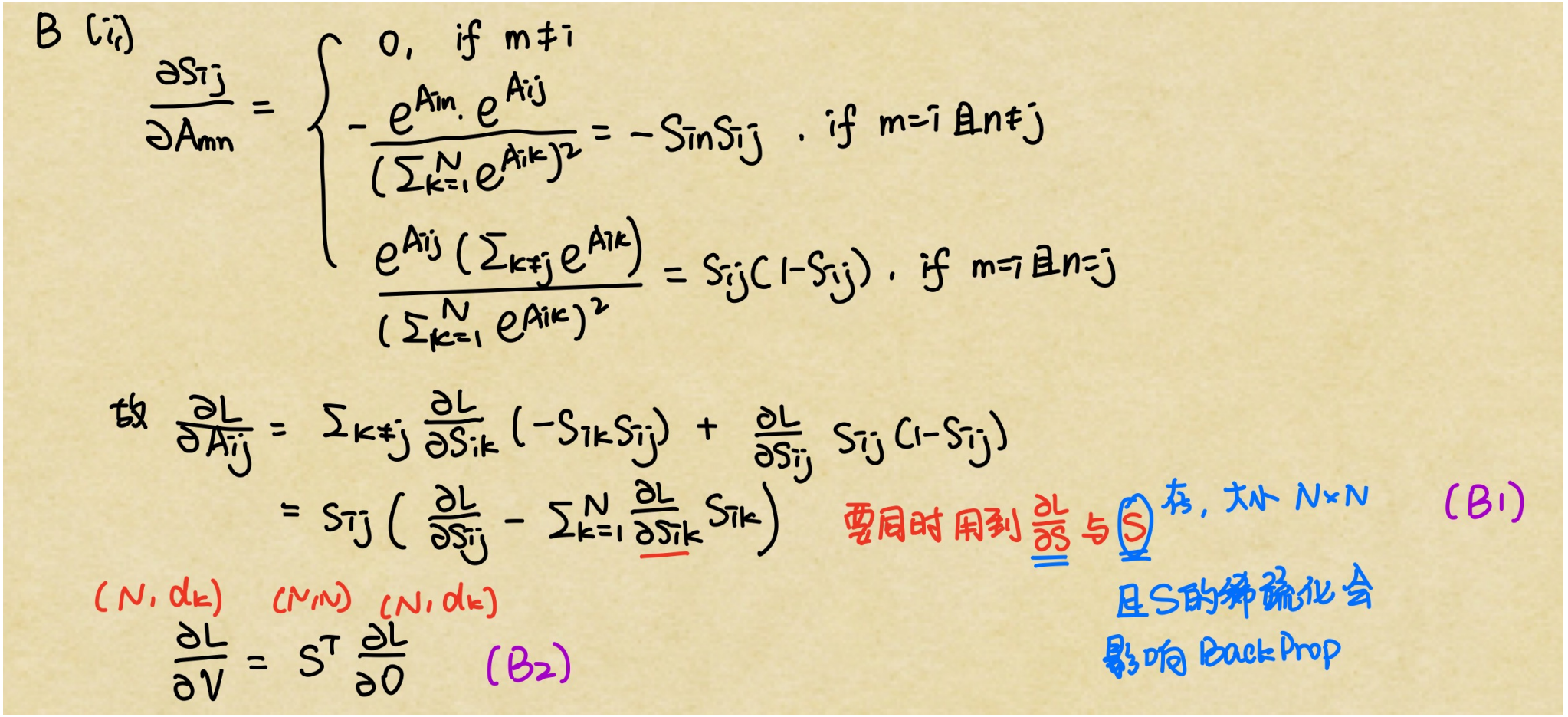
Save Q, K, V

Do not save X

Save S, V

Do not save A , O

Example: Attention



Experiments

北京大学 PEKING UNIVERSITY

While Block-wise training is memory-efficient, it converges slower than standard training.

Existing literature uses it to fine-tune, not to pretrain.

Fine-tune on a single RTX 3090

Method	Parameter	Gradient	Optimizer states	Memory consumption
Adam	16.1GB	32.1GB	96.6GB	144.8GB+
LOMO	16.1GB	0.5GB	—	21.5GB
LoRA-rank100	16.7GB	1.0GB	3.1GB	OOM
LoRA-rank8	16.2GB	0.1GB	0.3GB	22.3GB
Block-wise BAdam	16.1GB	0.9GB	2.6GB	23.5GB

[BAdam: A Memory Efficient Full Parameter Optimization Method for Large Language Models]

Experiments

北京大学 PEKING UNIVERSITY

Model: Llama 3-8B							
Method	GSM8K	Aqua	MMLU-Math	SAT-Math	MATH	NumGLUE	**Average**
Base model	25.9	22.8	33.7	39.5	12.8	34.5	28.2
Adam	**54.5**	40.5	44.3	51.4	**18.4**	55.4	44.1
LOMO	32.1	28.0	40.0	39.5	13.1	37.1	31.6
LoRA	47.5	**44.9**	45.3	50.9	14.5	**56.9**	43.3
Galore	33.1	37.4	41.2	42.7	15.0	36.9	34.4
BAdam	48.1	42.5	**50.5**	**56.8**	15.7	53.0	**44.4**
Model: Llama 3-70B							
Method	GSM8K	Aqua	MMLU-Math	SAT-Math	MATH	NumGLUE	**Average**
Base model	52.4	46.5	52.2	58.2	21.2	37.9	44.7
LoRA	73.3	59.5	58.3	64.1	**34.2**	64.8	59.0
BAdam	**78.8**	**63.4**	**64.2**	**76.4**	26.2	**67.3**	**62.7**

Table 6: Zero-shot math benchmark scores of the finetuned Llama 3-8B and Llama 3-70B on MathInstruct dataset by different methods.