北京大學
PEKING UNIVERSITY

Advanced LLM Optimizers

Yihan Jin, Ming Sun, Jiahe Geng,
Shuchen Zhu, Kun Yuan

Peking University

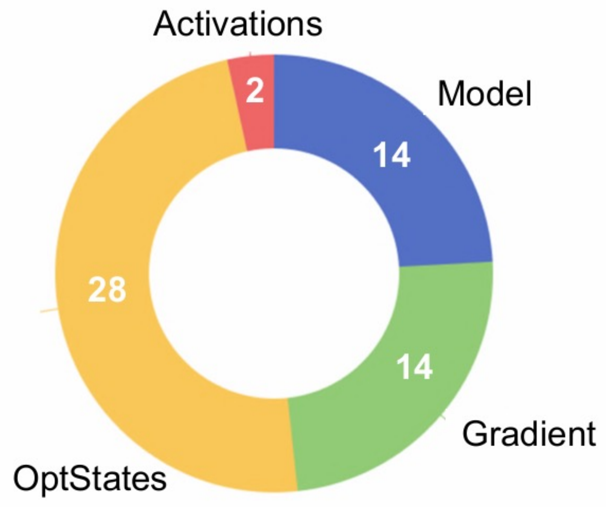
Recall: Memory Consumption in LLM Training

- Memory = **Model + Gradient + Optimizer States** + **Activation**
- Adam's Cost:

  Model Parameter – Φ

  Gradient - Φ

  Optimizer States (First & Second moment) - 2Φ



- Consequence:

  The 7B-pretrained model(BF16) requires 28GB of Adam optimizer states.

Goal: Decrease optimizer states while maintaining performance

Start with Adam: Destructing Adam's Redundancy

- Adam Update Rule:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \boldsymbol{\eta} \frac{\boldsymbol{M}_t}{\sqrt{\boldsymbol{V}_t} + \boldsymbol{\epsilon}}$$

where $\mathrm{W}_t \in \mathrm{R}^{\mathrm{m}\times\mathrm{n}}$ is weight matrix, $\mathrm{M}_\mathrm{t} \in \mathrm{R}^{\mathrm{m}\times\mathrm{n}}$ is **momentum**,

$\mathrm{V}_t \in \mathrm{R}^{\mathrm{m}\times\mathrm{n}}$ is second moment as **adaptive learning rate** (**Preconditioning**)

- Question:

1. Is it necessary to maintain an **adaptive learning rate** for each components in W_t?
2. If not, how should the **adaptability(V_t)** be arranged?

北京大学 PEKING UNIVERSITY

Adafactor: Adaptive Learning Rates with Sublinear Memory Cost

Noam Shazeer [1] **Mitchell Stern** [1 2]

[1]Google Brain, Mountain View, California, USA [2]University of California, Berkeley, California, USA. Correspondence to: Noam Shazeer <noam@google.com>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

Adafactor: Factorize V_t

北京大学 PEKING UNIVERSITY

- Assumption:

  $\mathrm{V}_t \in \mathrm{R}^{\mathrm{m}\times\mathrm{n}}$ is **low-rank**, i.e. $\mathrm{V}_t \approx R_t \cdot C_t$,

  where $R_t \in \mathrm{R}^{\mathrm{m}\times 1}$, $C_t \in \mathrm{R}^{1\times\mathrm{n}}$

- Motivation:

  Decrease the memory of optimizer states by **storing $\boldsymbol{R_t}$ & $\boldsymbol{C_t}$ instead of** V_t

- Memory Saving Results:

  O(mn) → O(m+n)

Adafactor: How to derive rank-1 factorization?

- Objective: Minimize **Generalized KL-Divergence** (I-Divergence)

$$D(V \,||\, RC) = \sum_{i,j}\left(V_{ij} log \frac{V_{ij}}{R_i C_j} - V_{ij} + R_i C_j\right)$$

- Constraints:

$$R_i \geq 0,\; C_j \geq 0$$

Adafactor: How to derive rank-1 factorization?

Theorem

The solution set of the optimization problem (minimizing I-divergence) when consists of all feasible pairs (R, S) satisfying:

$$RC = \frac{V 1_m 1_n^\top V}{1_n^\top V 1_m}$$

where $1_\ell = (1, \ldots, 1) \in \mathbb{R}^\ell$ *denotes a column vector of* ℓ *ones.*

- Solution (Closed-form):

$$R = V \cdot 1_m, \quad C = \frac{1_n^\top \cdot V}{1_n^\top \cdot V \cdot 1_m}$$

Proof of Theorem

$$D(V||RC) = \sum_{i=1}^{n}\sum_{j=1}^{m}\left(V_{ij}\log\frac{V_{ij}}{R_iC_j} - V_{ij} + R_iC_j\right)$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{m} V_{ij}\log V_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{m} V_{ij}\log R_i$$

$$- \sum_{i=1}^{n}\sum_{j=1}^{m} V_{ij}\log C_j - \sum_{i=1}^{n}\sum_{j=1}^{m} V_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{m} R_iC_j$$

Setting the derivatives with respect to R_i and C_j to 0:

$$\frac{\partial \mathcal{L}}{\partial R_i} = -\sum_{j=1}^{m}\frac{V_{ij}}{R_i} + \sum_{j=1}^{m} C_j = 0 \quad \Rightarrow \quad R_i = \frac{\sum_{j=1}^{m} V_{ij}}{\sum_{j=1}^{m} C_j}$$

$$\frac{\partial \mathcal{L}}{\partial C_j} = -\sum_{i=1}^{n}\frac{V_{ij}}{C_j} + \sum_{i=1}^{n} R_i = 0 \quad \Rightarrow \quad C_j = \frac{\sum_{i=1}^{n} V_{ij}}{\sum_{i=1}^{n} R_i}$$

Proof of Theorem

The solution has a scaling symmetry $(\alpha R, C/\alpha)$. We break this symmetry by enforcing the constraint $\sum_i R_i = \sum_{i,j} V_{ij}$.

This leads to the **canonical minimizer**:

$$R_i = \sum_{j=1}^{m} V_{ij}, \quad C_j = \frac{\sum_{i=1}^{n} V_{ij}}{\sum_{i=1}^{n} \sum_{j=1}^{m} V_{ij}}$$

In vector notation:

$$R = V\mathbb{1}_m, \quad C = \frac{\mathbb{1}_n^\top V}{\mathbb{1}_n^\top V \mathbb{1}_m}$$

Thus, the product RS is unique:

$$RC = V\mathbb{1}_m \left(\frac{\mathbb{1}_n^\top V}{\mathbb{1}_n^\top V \mathbb{1}_m} \right) = \frac{V\mathbb{1}_m \mathbb{1}_n^\top V}{\mathbb{1}_n^\top V \mathbb{1}_m} \qquad \square$$

Adafactor: Removing Momentum

- Motivation: Further save the memory of momentum – Φ
- Pseudo-code:

Algorithm 2 Adam for a matrix parameter X with factored second moments and first moment decay parameter $\beta_1 = 0$.

1: **Inputs:** initial point $X_0 \in \mathbb{R}^{n \times m}$, step sizes $\{\alpha_t\}_{t=1}^{T}$, second moment decay β_2, regularization constant ϵ
2: Initialize $R_0 = 0$ and $C_0 = 0$
3: **for** $t = 1$ **to** T **do**
4: $\quad G_t = \nabla f_t(X_{t-1})$
5: $\quad R_t = \beta_2 R_{t-1} + (1 - \beta_2)(G_t^2) 1_m$
6: $\quad C_t = \beta_2 C_{t-1} + (1 - \beta_2) 1_n^\top (G_t^2)$
7: $\quad \hat{V}_t = (R_t C_t / 1_n^\top R_t)/(1 - \beta_2^t)$
8: $\quad X_t = X_{t-1} - \alpha_t G_t / (\sqrt{\hat{V}_t} + \epsilon)$
9: **end for**

Rank-1 Factorization of V_t

Discard Momentum

Summary of Adafactor

- Pros:

  Less memory usage of optimizer state: 2mn $(M_t + V_t)$ → m+n $(R_t + C_t)$

- Cons:

  Approximation Error: V_t is not always of rank 1 → Slow convergence

  Throughput Cost: Factorize V_t and compute RMS increase **computation cost**

- Result:

  In order to achieve better performance, momentum M_t is **re-introduced** to the actual use of Adafactor, sacrificing memory to gain faster convergence speed.

ADAM-MINI: USE FEWER LEARNING RATES TO GAIN MORE

Yushun Zhang*[13], Congliang Chen*[13], Ziniu Li[13], Tian Ding[23], Chenwei Wu[4], Diederik P. Kingma[5], Yinyu Ye[16], Zhi-Quan Luo[13], Ruoyu Sun†[123]

[1] The Chinese University of Hong Kong, Shenzhen, China;
[2] Shenzhen International Center for Industrial and Applied Mathematics;
[3] Shenzhen Research Institute of Big Data; [4] Duke University;
[5] Anthropic; [6] Stanford University

Adam-mini: Start from Pre-conditioning Perspective

北京大學 PEKING UNIVERSITY

- Critique of Adafactor:

  It assumes $\boldsymbol{V_t}$ is rank-1. The assumption lacks **observation** to support.

  → Need to investigate NN's **structure**!

- View Adam as Pre-conditioning method:

$$w_{t+1} = w_t - \eta_t D_t m_t$$

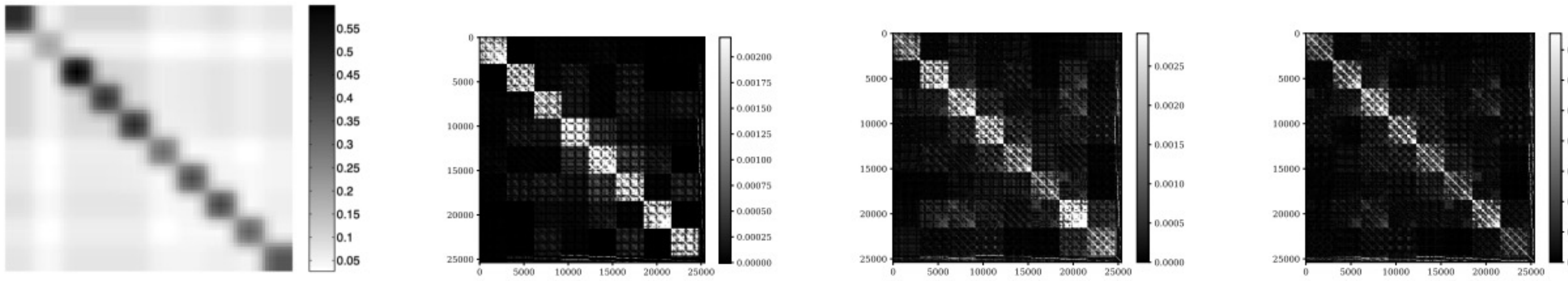
where $D_t = Diag\left(\frac{1}{\sqrt{v_t}}\right)$ is the pre-conditioning matrix

The ideal pre-conditioning matrix is Hessian's inverse H^{-1}

Need to inspect NN's Hessian structure!

Adam-mini: Hessian is near-block-diagonal

北京大学 PEKING UNIVERSITY



(a) Hessian of a MLP (Collobert, 2004)

(b) Hessian of a MLP at initialization

(c) Hessian of a MLP at 50% step

(d) Hessian of a MLP at 100% step

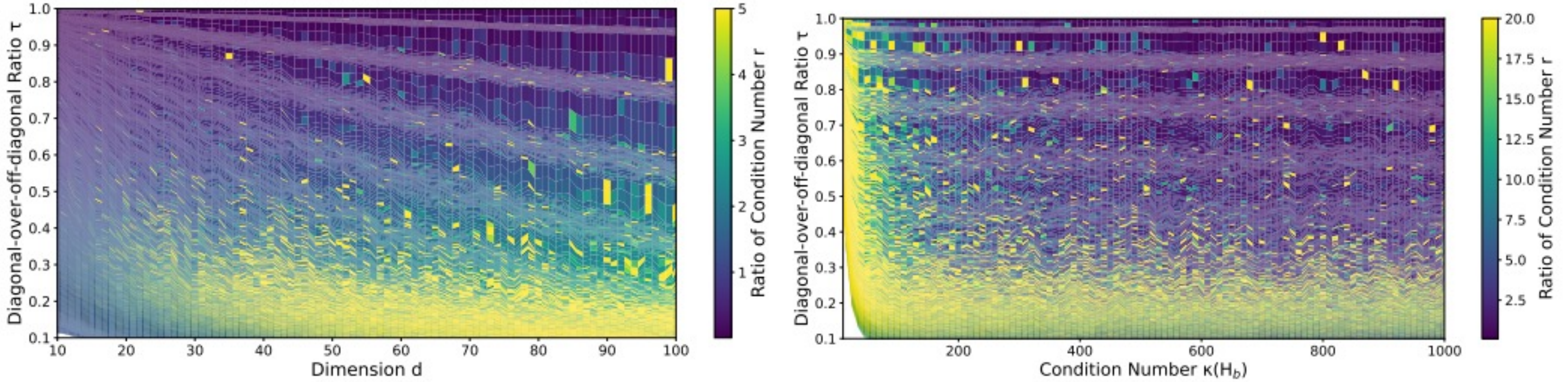
Figure 3: The near-block-diagonal Hessian structure of neural nets. (a) is the Hessian of an MLP after 1 training step reported in (Collobert, 2004). (b,c,d): the Hessians of a 1-hidden-layer MLP on CIFAR-100. The near-block-diagonal structure maintains throughout training, where each block corresponds to one neuron.

Adam-mini: How is Adam's pre-conditioning effect?

北京大学 PEKING UNIVERSITY

$$\tau = \frac{\sum_i |H_{b,i,i}|}{\sum_{i,j} |H_{b,i,j}|}$$: the degree of dominance of diagonal elements

$$r = \frac{\kappa(D_{Adam} H_b)}{\kappa(H_b)}$$: the pre-conditioning effect of Adam

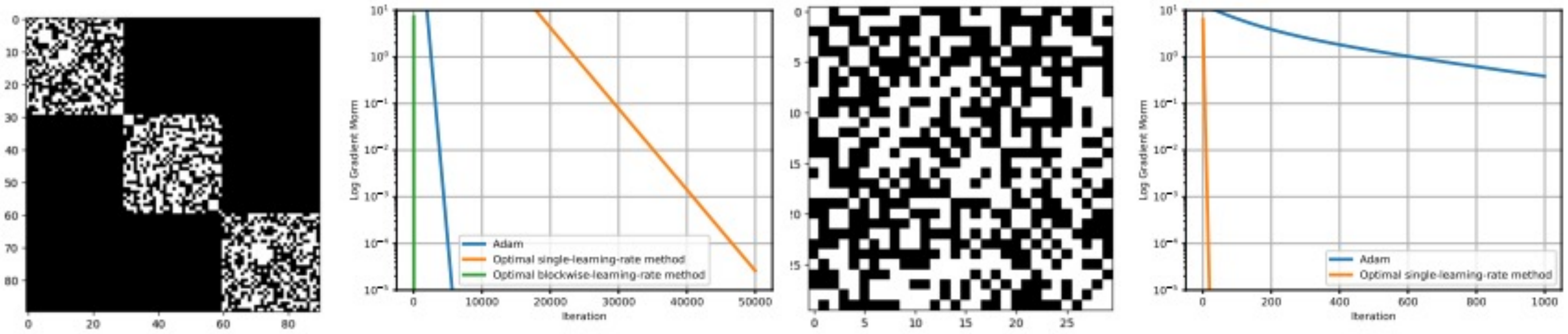


(a) r v.s. dimension d (b) r v.s. dimension $\kappa(H_b)$

Figure 5: The effectiveness of Adam's preconditioner D_{Adam} on different matrix structures of H_b. (a): for most dimension d, r is large when τ is small (r and τ are defined in Eq. (2)). This indicates that Adam might not be so effective when H_b is dense. We fix $\kappa(H_b) = 500$ here. (b): We use the same setups as (a), except that we fix the dimension $d = 50$ and change the x-axis to $\kappa(H_b)$.

Adam-mini: Case study of Adam's pre-conditioning effect

北京大学 PEKING UNIVERSITY



(a) Hessian matrix (b) Total loss (c) Dense sub-block (d) Sub-problem loss

Figure 4: **(a):** The Hessian of a three-block random quadratic problem. **(b):** Training curves for the problem associated with the full Hessian in (a). The optimal single (blockwise) learning rate is chosen based on the full (blockwise) Hessian in (a). **(c):** The 1st dense Hessian sub-blocks in (a). **(d):** Training curves for the new problem associated with the Hessian in (c).

Conclusion:

- For **dense Hessian case**, Adam is far inferior to **optimal single-learning-rate**.
- For block-diagonal Hessian case, Adam surpasses optimal single-learning-rate

Adam-mini: Observation Summary

- (Recall) Question:

  1. Is it necessary to maintain an adaptive learning rate for each components in W_t?

  2. If not, how should the adaptability($\boldsymbol{V_t}$) be arranged?

- For Q1:

  Under **near-block-diagonal Hessian structure**, Adam's maintaining an adaptive learning rate($\boldsymbol{V_t}$) for each components in W_t involves **redundancy**.

- For Q2:

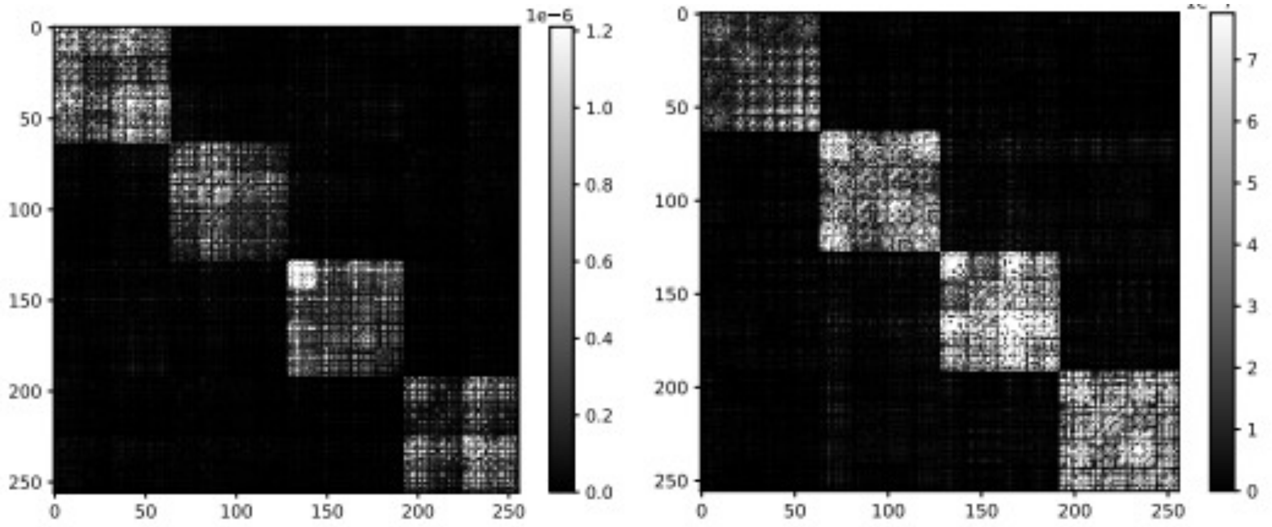
  For **each dense sub-block of Hessian**, carefully chosed **single learning rate** is good enough.

Adam-mini: Hessian based Transformer Partition Strategy

Use Hessian information to patition variables into groups:

- **Query/Key**: **Head-wise**

  Weight components in the same head as a block.



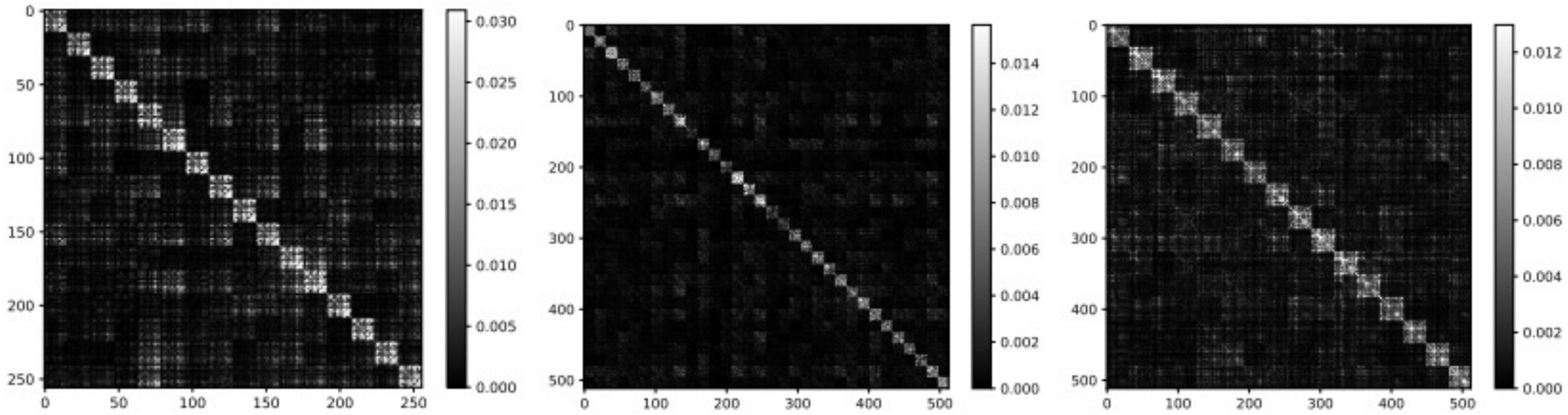
(a) query (4 heads) (b) key (4 heads)

Adam-mini: Hessian based Transformer Partition Strategy

Use Hessian information to patition variables into groups:

➢ **attn.proj/MLP**: **Neuron-wise**

Weight components in the same row as a block



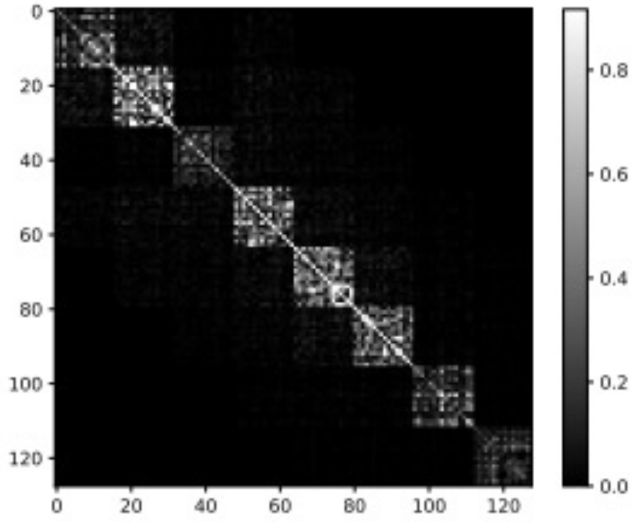
(d) `attn.proj` (16 neurons) (e) `mlp.fc_1` (32 neurons) (f) `mlp.proj` (16 neurons)

Adam-mini: Hessian based Transformer Partition Strategy

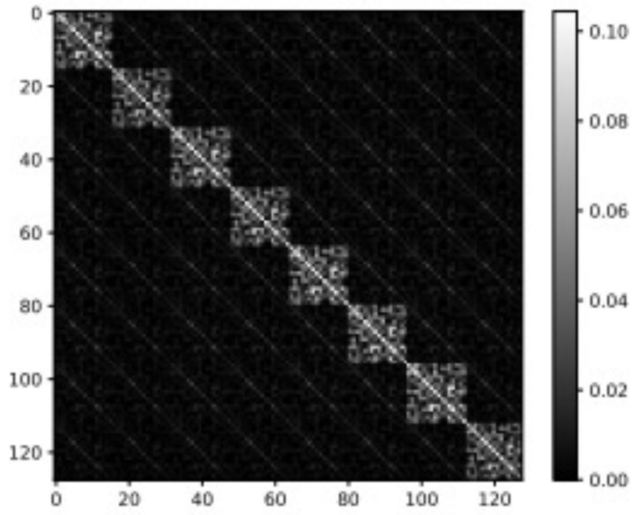
Use Hessian information to patition variables into groups:

- **Embedding/Output**: **Token-wise**

  Embedding – Each word vector as a block



(g) embed (8 tokens)



(h) output (8 tokens)

Adam-mini: How to set learning rate?

- For Adam: $u_{\text{Adam}} = \left(\frac{\eta}{\sqrt{v_1}}, \frac{\eta}{\sqrt{v_2}}, \frac{\eta}{\sqrt{v_3}}, \frac{\eta}{\sqrt{v_4}}, \frac{\eta}{\sqrt{v_5}}\right).$
- For Adam-mini: suppose the partition is $(1, 2, 3)$ and $(4, 5)$ then

$$u_{\text{mini}} = \left(\frac{\eta}{\sqrt{(v_1+v_2+v_3)/3}}, \frac{\eta}{\sqrt{(v_1+v_2+v_3)/3}}, \frac{\eta}{\sqrt{(v_1+v_2+v_3)/3}}, \frac{\eta}{\sqrt{(v_4+v_5)/2}}, \frac{\eta}{\sqrt{(v_4+v_5)/2}}\right).$$

➢ Why lr = mean(g $\odot$ g) ?

1. Convenience
2. Best among common statistics

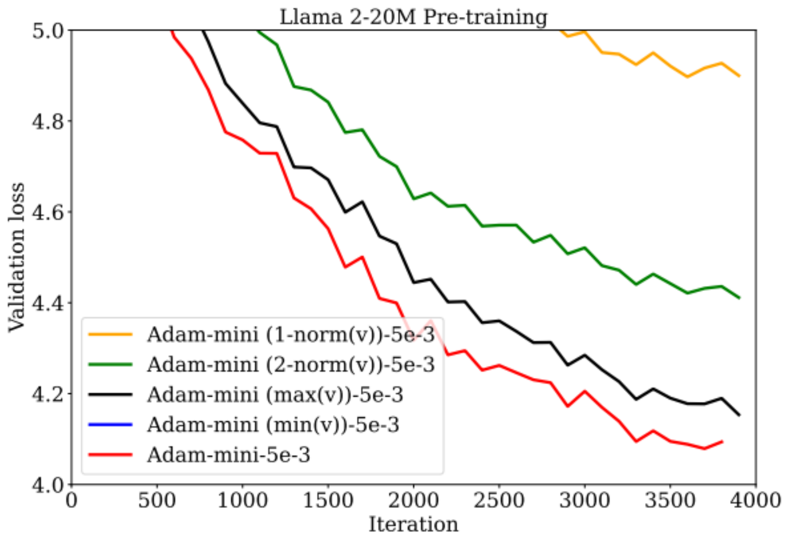


Figure 15: Ablation studies on the design of Adam-mini. We find that mean(v) performs better than other candidates. The blue curve does not show because the algorithm diverges and the curve is out of range.

Adam-mini

Algorithm 1 Adam-mini (General form)

Input weight-decay coefficient λ and current step t

Partition params into `param_blocks` by **Principle 1** in Section 2.3

for `param` in `param_blocks` **do**

  $\mathrm{g} = \mathtt{param.grad}$

  $\mathtt{param} = \mathtt{param} - \eta_t * \lambda * \mathtt{param}$

  $\mathrm{m} = (1-\beta_1) * \mathrm{g} + \beta_1 * \mathrm{m}$

  $\hat{\mathrm{m}} = \frac{\mathrm{m}}{1-\beta_1^t}$

  $\mathrm{v} = (1-\beta_2) * \mathtt{mean}(\mathrm{g} \odot \mathrm{g}) + \beta_2 * \mathrm{v}$

  $\hat{\mathrm{v}} = \frac{\mathrm{v}}{1-\beta_2^t}$

  $\mathtt{param} = \mathtt{param} - \eta_t * \frac{\hat{\mathrm{m}}}{\sqrt{\hat{\mathrm{v}}}+\epsilon}$

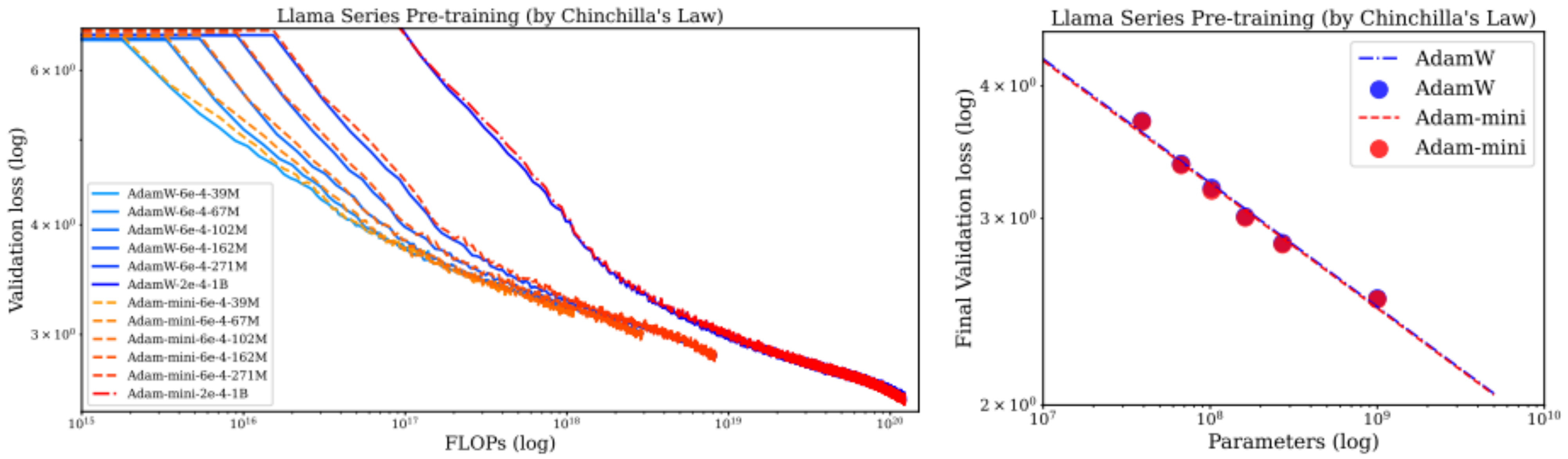
end for

Partition blocks based on Hessian

Single lr for a sub-block

Evaluation: Scaling Law

北京大学
PEKING UNIVERSITY



(a) Scaling laws in terms of compute

(b) Scaling laws in terms of parameters

Figure 11: (a, b): Scaling laws of Adam-mini. We pre-train Llama 2 architectures by Chinchilla's law. For all models sized from 39M to 1B, Adam-mini's loss curves are consistently similar to AdamW, but Adam-mini uses 50% less memory. Further, as shown in (b), Adam-mini reaches a lower final loss than AdamW for all models. The fitted lines in (b) suggest that Adam-mini can be scaled up to larger models (if the scaling law holds).

Adam-mini's loss curves closely resembles AdamW's

Adam-mini performs well using the same hyperparameter as AdamW

Evaluation: Memory & Throughput

Table 1: Memory cost of AdamW v.s. Adam-mini. Calculation is based on `float32`, which is a standard choice for optimizer states.

Model	Optimizer	Memory (GB)
GPT-2-1.5B	AdamW	12.48
GPT-2-1.5B	Adam-mini	6.24 (**50%** ↓)
Llama 2-1B	AdamW	8.80
Llama 2-1B	Adam-mini	4.40 (**50%** ↓)
Llama 2-7B	AdamW	53.92
Llama 2-7B	Adam-mini	26.96 (**50%** ↓)
Llama 3-8B	AdamW	64.24
Llama 3-8B	Adam-mini	32.12 (**50%** ↓)
Llama 2-13B	AdamW	104.16
Llama 2-13B	Adam-mini	52.08 (**50%** ↓)

Table 2: Throughput (↑) test on 2× A800-80GB GPUs for Llama 2-7B pre-training. ✗means out of memory. GPU hours (↓) to pre-train Llama 2-7B with the optimal token amount by Chinchila's law.

Optimizer	bs_per_GPU	total_bs	Throughput (↑)
Adam-mini	4	256	5572.19 (↑ **49.6%**)
AdamW	2	256	✗
AdamW	1	256	3725.59

Optimizer	# Tokens (B)	GPU hours (h) (↓)
AdamW	1	74.56
Adam-mini	1	49.85 (↓ **33.1%**)
AdamW	70	5219.16
Adam-mini	70	3489.55 (↓ **33.1%**)
AdamW	140	10438.32
Adam-mini	140	6979.10 (↓ **33.1%**)

Compared to AdamW, Adam-mini saves 50% memory, has 49.6% higher throughput.

Summary of Adam-mini

Efficiency:

- **Memory**: Less memory usage of optimizer state: 2mn $(M_t + V_t) \rightarrow$ mn (M_t)
- **Hyperparameter**: Performs well using the same hyperparameter as AdamW
- **Computation**: Substitute vector operations like *sqrt* & *div* by scalar operation

Second-order Methods

- Non-constrained Optimization Problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$$

- Taylor's Formula:

$$f(\mathbf{x}_k+\mathbf{p}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \mathbf{H}(\mathbf{x}_k)\mathbf{p}$$

- Update Vector:

$$\mathbf{p}_k^{\text{Newton}} = -\mathbf{H}_k^{-1}\nabla f(\mathbf{x}_k)$$

Second-order Methods

➢ Advantage of Second-order Methods: Curvature Calibration

Eigen Value Decomposition:

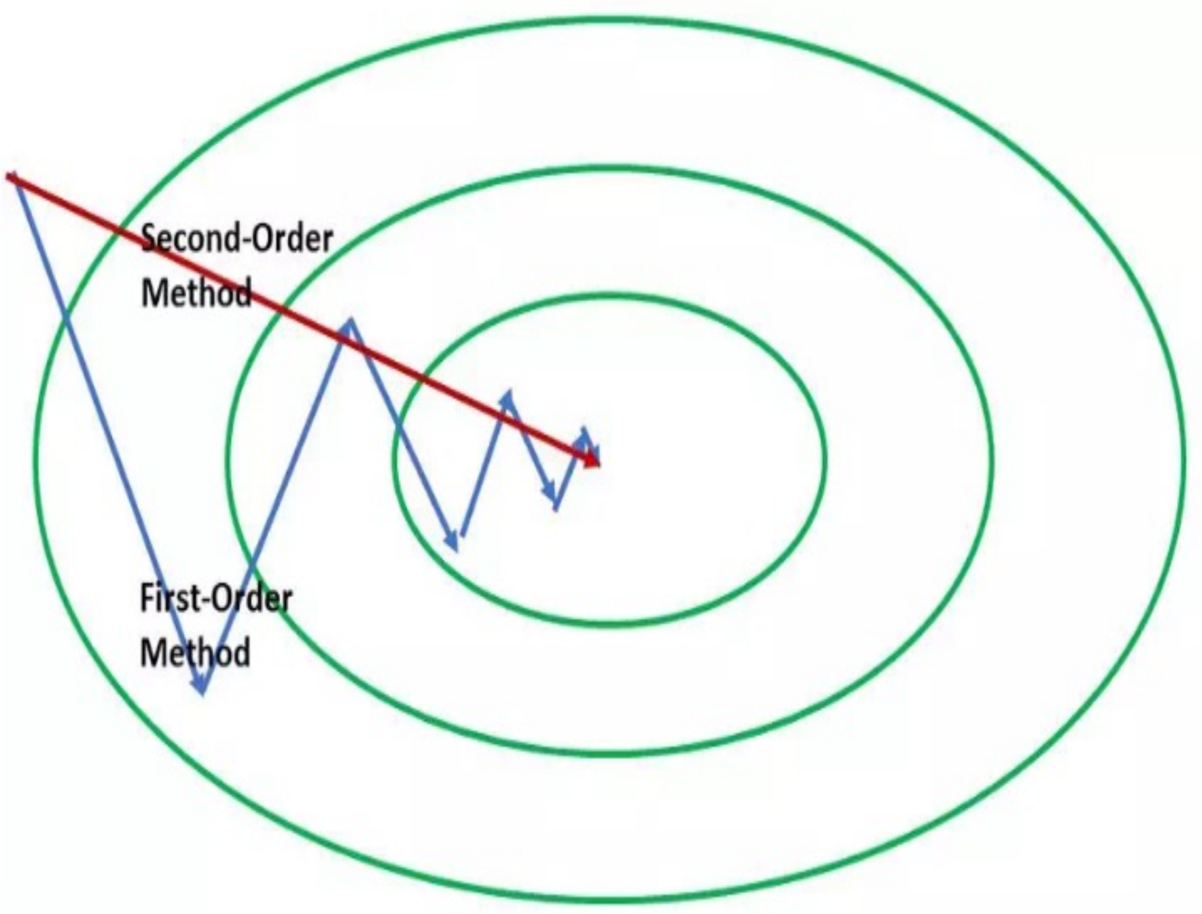
$$\mathbf{H}_k = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$$

Then we have:

$$\mathbf{p}_k^{\text{Newton}} = -\mathbf{Q}\mathbf{\Lambda}^{-1}\mathbf{Q}^\top \nabla f(\mathbf{x}_k)$$

In the Eigenspace:

$$\tilde{\mathbf{p}} = -\mathbf{\Lambda}^{-1}\tilde{\nabla} f$$



Second-order Methods

- However, it is always hard to get Hessian Matrix
- For Weight matrix $W \in \mathbb{R}^{m \times n}$, the Hessian Matrix is:

$$H \in \mathbb{R}^{mn \times mn}$$

Computational complexity for H^{-1}: $O(m^3 n^3)$

- For LLM with 1B parameters, the memory cost for Hessian Matrix is:

$$10^{18}\ elements \times 8\ bytes/element = 8 \times 10^{18}\ bytes = 10^6 TB$$

The computational complexity for H^{-1} is: $10^{27}\ FLOP$

Second-order Methods

- Can we decompose the Hessian Matrix?

- Kronecker Product: Given two matrices: $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{p \times q}$,

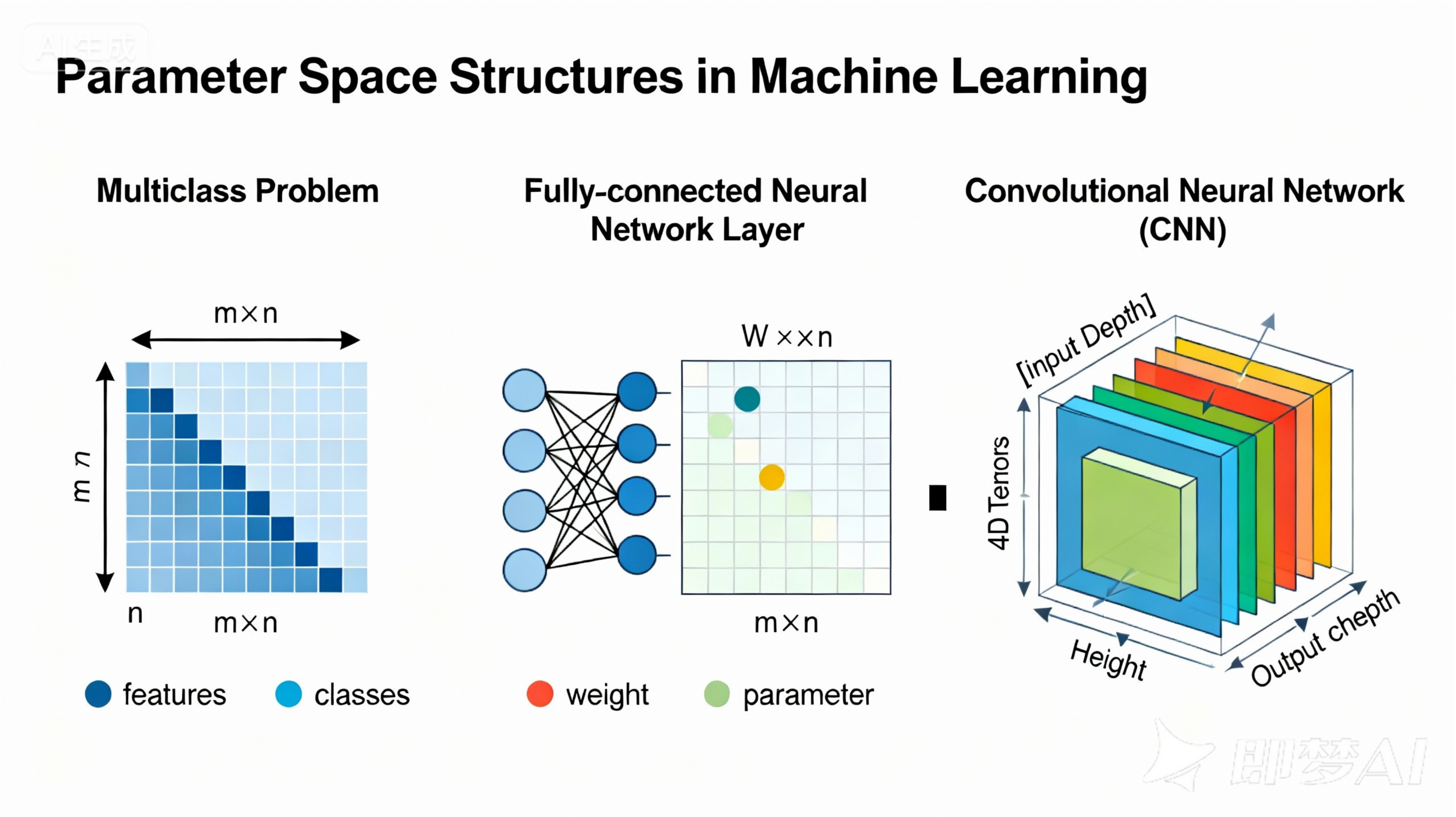
The **Kronecker product** of A and B, denoted $A \otimes B$, is defined as the block matrix:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$$

The resulting matrix $A \otimes B$ has dimensions $(mp) \times (nq)$.

Shampoo

➢ Motivation: Exploits the structure of the parameter space



Shampoo

	Memory Cost	Computational Cost
➢ Vector-form Second-order Method:		
$\boldsymbol{W_t = W_{t-1} - \eta_t H_t^{-1} G_t}$	$O(m^2n^2)$	$O(m^3n^3)$
➢ Shampoo：		
$\boldsymbol{L}_t = \beta \boldsymbol{L}_{t-1} + \boldsymbol{G}_t \boldsymbol{G}_t^\top$	$O(m^2)$	$O(m^2n)$
$\boldsymbol{R}_t = \beta \boldsymbol{R}_{t-1} + \boldsymbol{G}_t^\top \boldsymbol{G}_t$	$O(n^2)$	$O(mn^2)$
$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta_t \boldsymbol{L}_t^{-1/4} \boldsymbol{G}_t \boldsymbol{R}_t^{-1/4}$	$O(mn)$	$O(m^3 + n^3 + m^2n + mn^2)$

Shampoo

$$F \approx \mathbb{E}\left[gg^T\right]$$

$$g = \mathrm{vec}(G) \in \mathbb{R}^{mn}$$

$$F = \mathbb{E}\left[gg^T\right] = \mathbb{E}\left[\mathrm{vec}(G)\,\mathrm{vec}(G)^T\right]$$

Shampoo

$$G = \delta x^T \quad \in \mathbb{R}^{m \times n}$$

利用kronecker乘积的性质

$$F = \mathbb{E}\left[\operatorname{vec}(G)\operatorname{vec}(G)^T\right] = \mathbb{E}\left[\left(xx^T\right) \otimes \left(\delta\delta^T\right)\right]$$

认为输入 x 和误差信号 δ 的联合统计可以近似为“独立”

$$F \approx \mathbb{E}\left[xx^T\right] \otimes \mathbb{E}\left[\delta\delta^T\right]$$

Shampoo

$$\mathbb{E}\left[GG^T\right] = \mathbb{E}\left[||x||^2\delta\delta^T\right] \qquad \mathbb{E}\left[G^TG\right] = \mathbb{E}\left[||\delta||^2 xx^T\right]$$

$||x||^2$、$||\delta||^2$ 与方向统计可以分离

$$L_t = \beta L_{t-1} + (1-\beta)G_tG_t^T$$

$$R_t = \beta R_{t-1} + (1-\beta)G_t^TG_t$$

Shampoo

$$(L \otimes R)^{-1} = L^{-1} \otimes R^{-1}$$

$$\mathrm{vec}\left(AXB^T\right) = (B \otimes A)\,\mathrm{vec}(X)$$

$$\Downarrow$$

$$H^{-1}\,\mathrm{vec}(G) = (L \otimes R)^{-1}\,\mathrm{vec}(G) = \mathrm{vec}(L^{-1}GR^{-1})$$

$$\Downarrow$$

$$\text{Update Matrix } = L^{-1} \cdot G \cdot R^{-1}$$

Shampoo

➢ Algorithm in matrix case & tensor case

Algorithm 1 Shampoo, matrix case.

Initialize $W_1 = \mathbf{0}_{m\times n}$; $L_0 = \epsilon I_m$; $R_0 = \epsilon I_n$

for $t = 1, \ldots, T$ **do:**

  Receive loss function $f_t : \mathbb{R}^{m\times n} \mapsto \mathbb{R}$

  Compute gradient $G_t = \nabla f_t(W_t)$ // $G_t \in \mathbb{R}^{m\times n}$

  Update preconditioners:

$$L_t = L_{t-1} + G_t G_t^{\top}$$

$$R_t = R_{t-1} + G_t^{\top} G_t$$

  Update parameters:

$$W_{t+1} = W_t - \eta L_t^{-1/4} G_t R_t^{-1/4}$$

Algorithm 2 Shampoo, general tensor case.

Initialize: $W_1 = \mathbf{0}_{n_1\times\cdots\times n_k}$; $\forall i \in [k] : H_0^i = \epsilon I_{n_i}$

for $t = 1, \ldots, T$ **do:**

  Receive loss function $f_t : \mathbb{R}^{n_1\times\cdots\times n_k} \mapsto \mathbb{R}$

  Compute $G_t = \nabla f_t(W_t)$ // $G_t \in \mathbb{R}^{n_1\times\cdots\times n_k}$

  $\widetilde{G}_t \leftarrow G_t$ // $\widetilde{G}_t$ *is preconditioned gradient*

  for $i = 1, \ldots, k$ **do:**

   $H_t^i = H_{t-1}^i + G_t^{(i)}$

   $\widetilde{G}_t \leftarrow \widetilde{G}_t \times_i (H_t^i)^{-1/2k}$

  Update: $W_{t+1} = W_t - \eta \widetilde{G}_t$

Shampoo

➢ Experiment Results

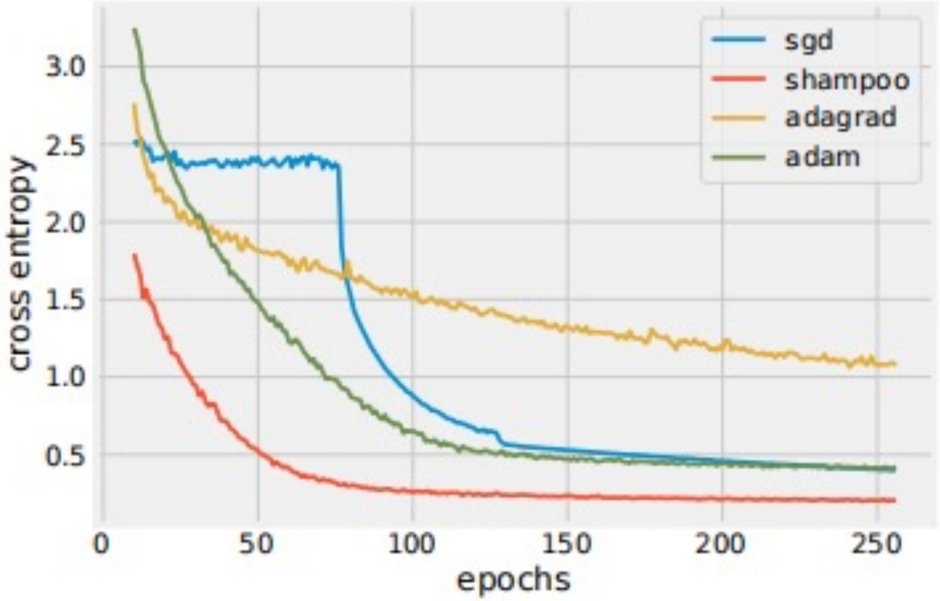


Figure 3. Convergence of training loss for a 55-layer ResNet on CIFAR-100.

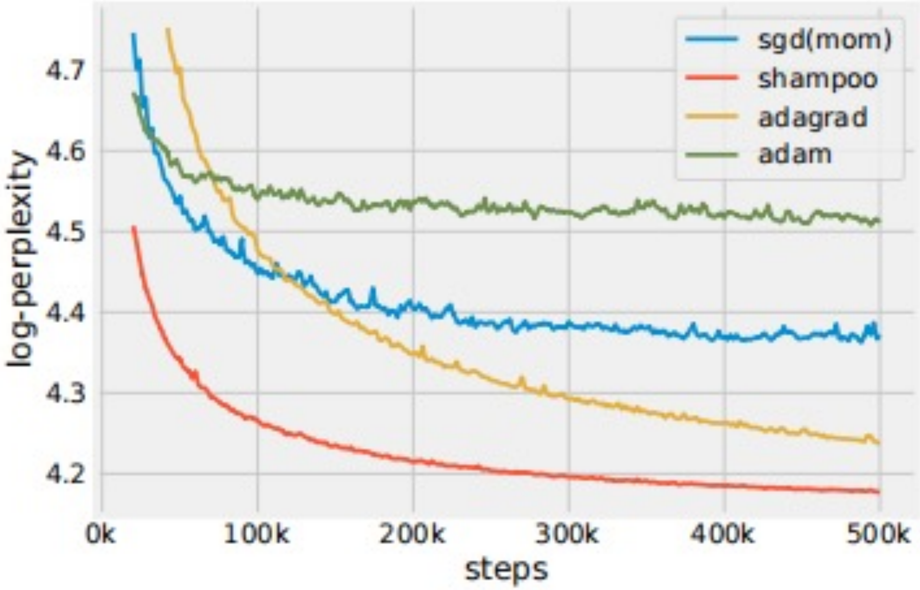
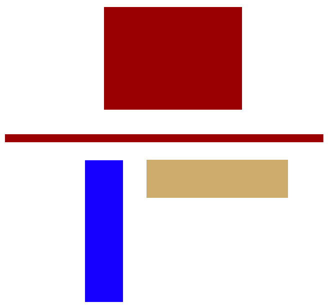


Figure 4. Convergence of test loss for the Transformer model for machine translation (Vaswani et al., 2017) on LM1B.

Essence of Shampoo

- Eigen Decomposition：



$$L = Q_L \Lambda_L Q_L^T$$

$$\text{Update} = Q_L(\Lambda_L^{-1/4} \underbrace{Q_L^T G Q_R}_{\text{旋转后的梯度 } G'} \Lambda_R^{-1/4}) Q_R^T$$

$$\Downarrow$$

$$\Lambda_L^{-1/4} \cdot G' \cdot \Lambda_R^{-1/4} = \frac{G'}{\text{diag}\left(\Lambda_L^{\frac{1}{4}}\right) \text{diag}\left(\Lambda_R^{\frac{1}{4}}\right)^T}$$

SOAP

PEKING UNIVERSITY 北京大学

- Observation: The variant of Shampoo is equivalent to running Adafactor in the eigenbasis provided by Shampoo's preconditioner

Algorithm 1 Single step of idealized Shampoo with power $1/2$.

1: Sample batch B_t.
2: $G_t \in \mathbb{R}^{m\times n} \leftarrow -\nabla_W \phi_{B_t}(W_t)$
3: $L \leftarrow \mathbb{E}_B[G_B G_B^T]$ {Where the expectation is over a random batch B.}
4: $R \leftarrow \mathbb{E}_B[G_B^T G_B]$
5: $\hat{H} \leftarrow L \otimes R/\text{Trace}(L)$
6: $W_t \leftarrow W_{t-1} - \eta \hat{H}^{-1/2} G_t = W_{t-1} - \eta L^{-1/2} G_t R^{-1/2}/\text{Trace}(L)^{-1/2}$

Algorithm 2 Single step of idealized Adafactor in Shampoo's eigenspace.

1: Sample batch B_t.
2: $G_t \in \mathbb{R}^{m\times n} \leftarrow -\nabla_W \phi_{B_t}(W_t)$
3: $L \leftarrow \mathbb{E}_B[G_B G_B^T]$
4: $R \leftarrow \mathbb{E}_B[G_B^T G_B]$
5: $Q_L \leftarrow \texttt{Eigenvectors}(L)$
6: $Q_R \leftarrow \texttt{Eigenvectors}(R)$
7: $G'_t \leftarrow Q_L^T G_t Q_R$
8: {Idealized version of code for Adafactor taking G'_t to be the gradient}
9: $G'_B \leftarrow Q_L^T G_B Q_R$
10: $A = \mathbb{E}_B[G'_B \odot G'_B]\mathbf{1}_m$ where $G'_B = Q_L^T G_B Q_R$
11: $C = \mathbf{1}_n^\top \mathbb{E}_B[G'_B \odot G'_B]$
12: $\hat{V}_t = \frac{AC^T}{\mathbf{1}_n^\top A}$ {Elementwise division}
13: $G''_t \leftarrow \frac{G'_t}{\sqrt{\hat{V}_t + \epsilon}}$ {Elementwise division and square root}
14: $G'''_t \leftarrow Q_L G''_t Q_R^T$ {Projecting back to original space}
15: $W_t \leftarrow W_{t-1} - \eta G'''_t$

SOAP

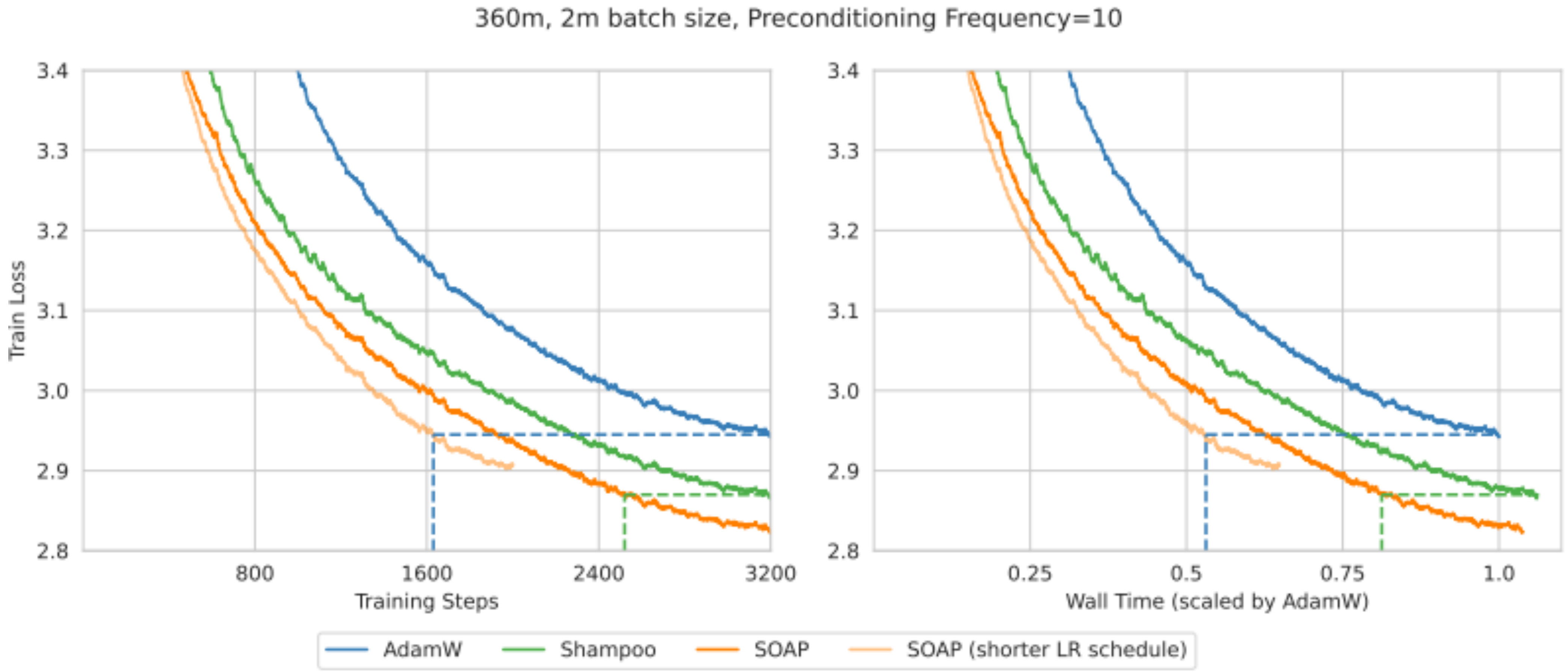
- Inspiration: a broader design space for combining first and second order methods——running a first-order method in the eigenbasis provided by a second-order method

Algorithm 3 Single step of SOAP for a $m \times n$ layer. Per layer, we maintain four matrices: $L \in \mathbb{R}^{m\times m}$, $R \in \mathbb{R}^{n\times n}$ and $V, M \in \mathbb{R}^{m\times n}$. For simplicity we ignore the initialization and other boundary effects such as bias correction. Hyperparameters: Learning rate η, betas $= (\beta_1, \beta_2)$, epsilon ϵ, and preconditioning frequency f.
An implementation of SOAP is available at https://github.com/nikhilvyas/SOAP/tree/main.

Sample batch B_t.
$G \in \mathbb{R}^{m\times n} \leftarrow -\nabla_W \phi_{B_t}(W_t)$
$G' \leftarrow Q_L^T G Q_R$
$M \leftarrow \beta_1 M + (1-\beta_1)G$
$M' \leftarrow Q_L^T M Q_R$
{Now we "run" Adam on G'}
$V \leftarrow \beta_2 V + (1-\beta_2)(G' \odot G')$ {Elementwise multiplication}
$N' \leftarrow \frac{M'}{\sqrt{\hat{V}_t}+\epsilon}$ {Elementwise division and square root}
{Now that we have preconditioned by Adam in the rotated space, we go back to the original space.}
$N \leftarrow Q_L N' Q_R^T$
$W \leftarrow W - \eta N$
{End of gradient step, we now update L and R and possibly also Q_L and Q_R. }
$L \leftarrow \beta_2 L + (1-\beta_2) G G^T$
$R \leftarrow \beta_2 R + (1-\beta_2) G^T G$
if t % f == 0 **then**
    $Q_L \leftarrow \texttt{Eigenvectors}(L, Q_L)$
    $Q_R \leftarrow \texttt{Eigenvectors}(R, Q_R)$
end if

SOAP

➤ Better robustness, Faster training



Muon: An Orthogonalization-Based Optimizer for Deep Networks

Keller Jordan

Motivation: Limitations of Standard Optimizers

- **Standard view:** parameters in deep learning are a long vector; we use SGD/Adam/AdamW on this vector. But hidden layers are actually matrices.

- **Empirical issue:**
  - Gradients/updates often have highly skewed singular values → poor conditioning
  - Many directions updated very weakly → slow learning of rare / subtle patterns

- **Question:** can we design an optimizer that respects **matrix structure** and **improves conditioning**?

Muon Update Rules:

- Outline:

$$G_t \xrightarrow{\text{momentum}} M_t \xrightarrow{-\eta} U_t \xrightarrow{\text{NS-ortho}} Q_t \xrightarrow{+\alpha} W_{t+1}.$$

- Details:

$$\begin{aligned}
\textbf{Gradient:}\quad & G_t = \nabla_W L(W_t),\\
\textbf{Momentum:}\quad & M_t = \beta M_{t-1} + (1-\beta)\, G_t,\\
\textbf{Raw update:}\quad & U_t = -\eta\, M_t,\\
\textbf{Orthogonalization:}\quad & Q_t = \mathrm{Ortho}_{\mathrm{NS}}(U_t),\\
\textbf{Parameter update:}\quad & W_{t+1} = W_t + \alpha\, Q_t.
\end{aligned}$$

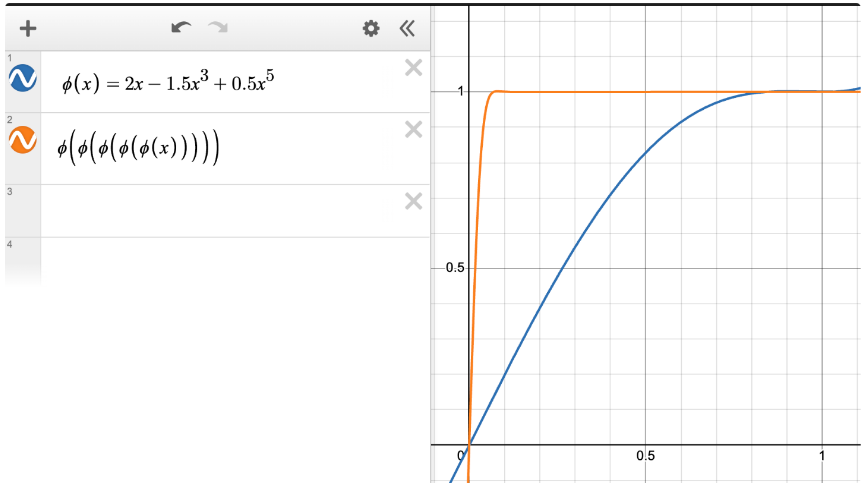
Orthogonalization via K-step Newton–Schulz:

- Newton–Schulz iteration performs a **fast** and **low-cost approximate orthogonalization** of the target matrix. (SVD for parameter matrix is expensive.)
- How it works:

$$\textbf{Initialization:} \quad U^{(0)} = \frac{U_t}{\|U_t\|_F},$$

$$\textbf{Iterations:} \quad U^{(k+1)} = a\,U^{(k)} + b\,U^{(k)}{U^{(k)}}^\top U^{(k)} + c\,(U^{(k)}{U^{(k)}}^\top)^2 U^{(k)}, \quad k = 0, \ldots, K-1,$$

$$\textbf{Output:} \quad Q_t = U^{(K)} \approx \mathrm{Ortho}_{\mathrm{NS}}(U_t).$$



- (a,b,c)=(3.1415,4.7750,2.0315) ,N=5 for final Muon design.

Why is it good to orthogonalize the update?

➢ **What does orthogonalization in Muon do:**

Let $W_t \in \mathbb{R}^{n \times m}$ be the weight matrix at time step t, and G_t be the gradient of the loss function with respect to W_t:

$$G_t = \nabla_W L(W_t)$$

The key idea is to apply an orthogonalization operator to U_t. The polar decomposition of a matrix G is given by:

$$G = QP$$

where Q is a semi-orthogonal matrix and P is a symmetric positive semidefinite matrix. The matrix Q is the nearest orthogonal matrix to G.

$$Q = \arg \min_{O : O^\top O = I} \|O - G\|_F$$

where O is any semi-orthogonal matrix. This is equivalent to:

$$Q = G(G^\top G)^{-\frac{1}{2}}$$

Why is it good to orthogonalize the update?

北京大学 PEKING UNIVERSITY

- **Properties of orthogonalization:**
  - Orthogonalization forces the **singular values** of $\widetilde{U}_t$ to be equal to 1, which improves the **conditioning** of the update.

$$\text{Singular values of } \widetilde{U}_t : \quad \sigma_i(\widetilde{U}_t) = 1 \quad \forall i$$

- **Why is it good to orthogonalize the update?**
  - Updates produced by both SGD-momentum and Adam for the 2D parameters in transformer-based neural networks typically are almost **low-rank matrices,** with the updates for all neurons being dominated by just a few directions.
  - Orthogonalization effectively increases the scale of other **"rare directions"** which have small magnitude in the update but are nevertheless **important for learning**.

Runtime Analysis of Muon

➢ NS Iteration and Extra FLOPs:

- Before the NS iteration is applied, Muon is just a standard SGD-momentum optimizer, so it has the same memory requirement.
- For each $n \times m$ matrix parameter in the network, each step of the NS iteration requires $2(2nm^2+m^3)$ matmul FLOPs.
- Therefore, the extra FLOPs required by Muon compared to SGD is at most $2T(2nm^2+m^3)$, where T is the number of NS iterations (typically $T=5$).

➢ Extra computation rate:

- If the parameter parametrizes a linear layer, then the baseline amount of FLOPs used to perform a training step (i.e., a forward and backward pass) is $6nmB$, where B is the batch size in tokens.
- Therefore, the FLOP overhead of Muon is at most $\boxed{Tm/B,}$ where m is the model dimension, B is the batch size in tokens, and T is the number of NS iteration steps

Runtime Analysis of Muon

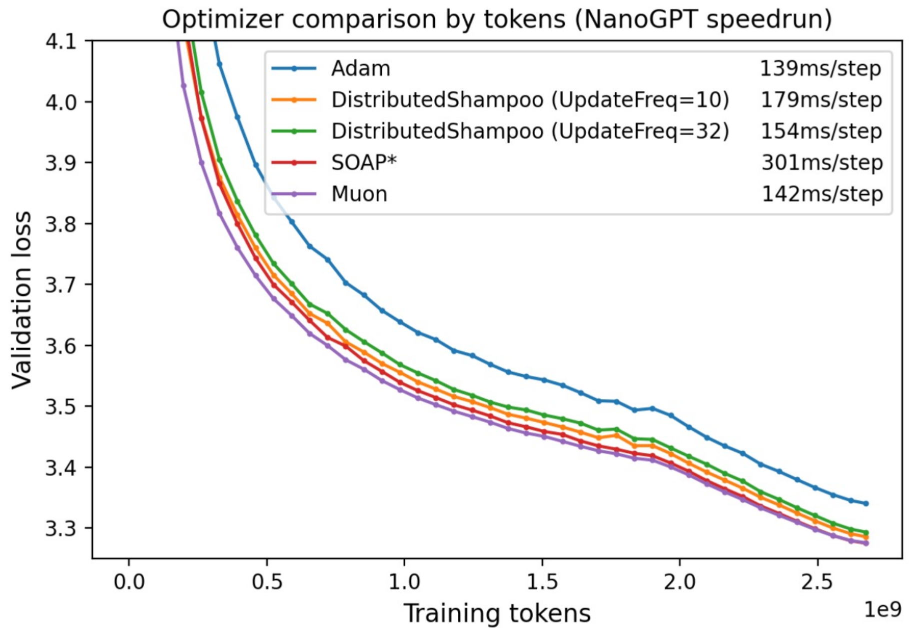
NanoGPT overhead of using Muon:

- For the current NanoGPT speedrunning record, the model dimension is $m = 768$ and the number of tokens per batch is $B = 524288$.
- Therefore, the overhead of using Muon is $\frac{5\times 768}{524288} = \boxed{0.7\%.}$

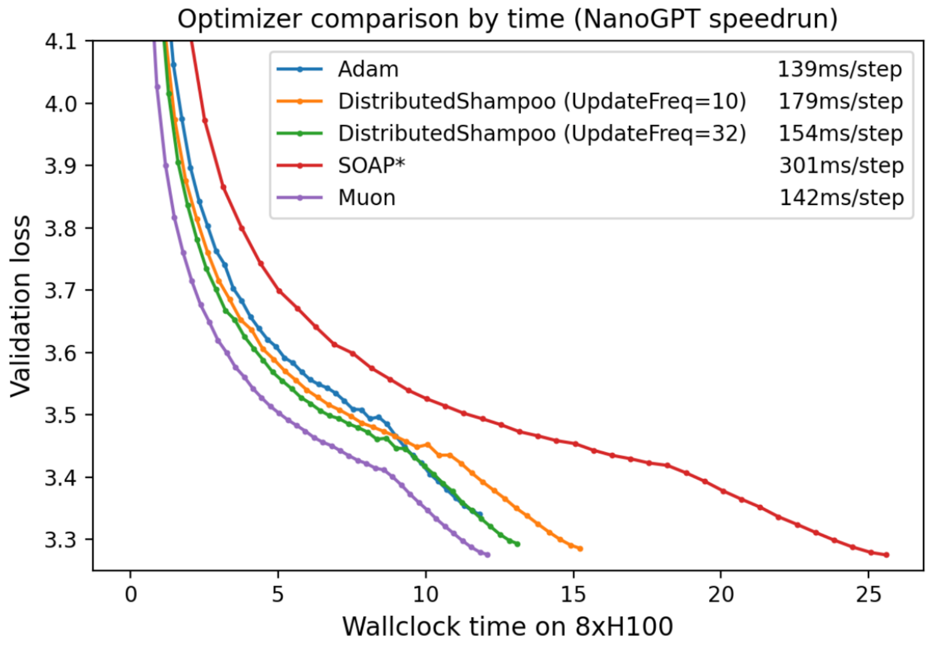
Llama 405B overhead of using Muon:

- For Llama 405B training, the model dimension is $m = 16384$ and the number of tokens per batch is reported to be $B = 16000000$.
- Therefore, the overhead of using Muon for this training would be $\frac{5\times 16384}{16000000} = \boxed{0.5\%.}$

Muon Empirical Results (Jordan)



*SOAP is under active development. Future versions will significantly improve the wallclock overhead.



*SOAP is under active development. Future versions will significantly improve the wallclock overhead.

Fig1: Improved the speed record for training to 3.28 val loss on FineWeb (a competitive task known as NanoGPT speedrunning) by a factor of **1.35x.**

Fig2: Optimizer comparison by wallclock time.

Muon Empirical Results (Kimi)

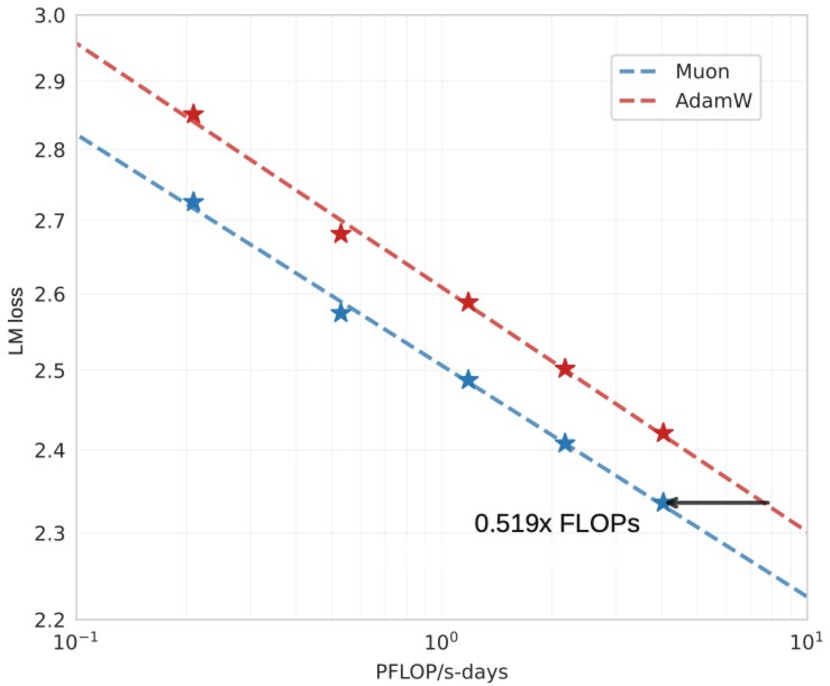


Table 4: Comparison of different models at around 1.2T tokens.

	Benchmark (Metric)	**DSV3-Small**	**Moonlight-A@1.2T**	**Moonlight@1.2T**
	Activated Params†	2.24B	2.24B	2.24B
	Total Params†	15.29B	15.29B	15.29B
	Training Tokens	1.33T	1.2T	1.2T
	Optimizer	AdamW	AdamW	Muon
English	MMLU	53.3	60.2	**60.4**
	MMLU-pro	-	26.8	**28.1**
	BBH	41.4	**45.3**	43.2
	TriviaQA	-	57.4	**58.1**
Code	HumanEval	26.8	29.3	**37.2**
	MBPP	36.8	49.2	**52.9**
Math	GSM8K	31.4	43.8	**45.0**
	MATH	10.7	16.1	**19.8**
	CMath	-	57.8	**60.2**
Chinese	C-Eval	-	57.2	**59.9**
	CMMLU	-	58.2	**58.8**

† The reported parameter counts exclude the embedding parameters.

Fig1: Muon uses ~52% less computational cost (FLOPs) during training compared to Adam optimizer in **Llama architechture**.

Table4: Moonlight(trained by Muon) performs significantly better than Moonlight-A(trained by AdamW), proving the scaling effectiveness of Muon. We observed that Muon especially excels on **Math and Code** related tasks.

北京大學
PEKING UNIVERSITY

Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training

Hong Liu Zhiyuan Li David Hall Percy Liang Tengyu Ma

Stanford University

{hliu99, zhiyuanli, dlwh, pliang, tengyuma}@cs.stanford.edu

Motivations: Challenges in Large-Scale LLM Training

- AdamW uses only **first-order gradients**, while Loss landscape is **highly anisotropic.**
  - **Some directions: high curvature**
  - **Others: flat**
- **Uniform step sizes cause:**
  - **Instability in steep directions**
  - **Slow progress in flat directions**

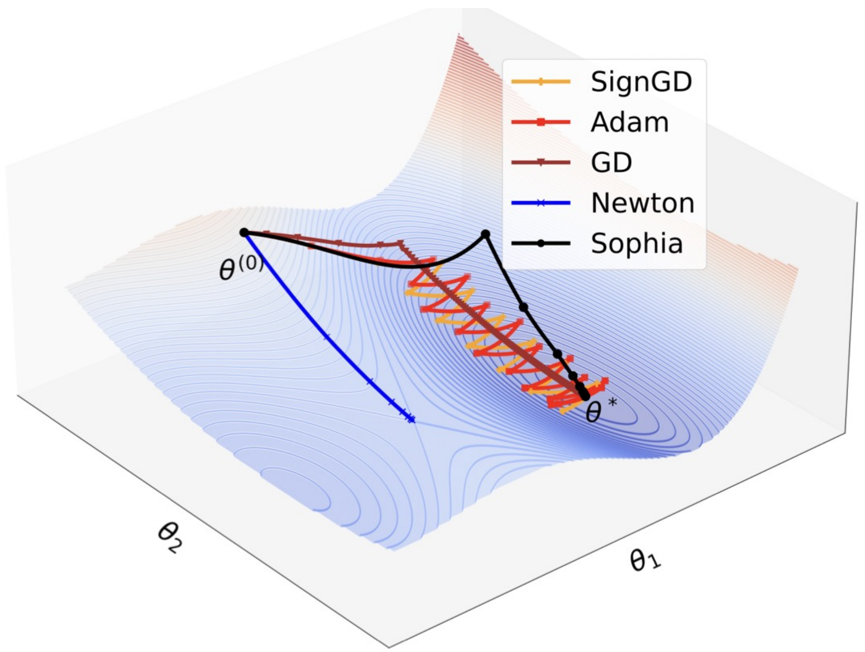


Fig: The motivating toy example. θ[1] is the sharp dimension and θ[2] is the flat dimension. GD's learning rate is limited by the sharpness in θ[1], and makes slow progress along θ[2]. Adam and SignGD bounce along θ[1] while making slow progress along θ[2]. Vanilla Newton' s method converges to a saddle point. Sophia makes fast progress in both dimensions and converges to the minimum with a few steps.

Motivations: Need for Second-Order Information

- Second-order curvature helps scale updates per coordinate, but Full Hessian is impossible to compute or store.
  - Size = $O(n^2)$ for n parameters.
  - Inverting or factorizing Hessian is infeasible.
- Goal: Use curvature approximation that is:
  - Informative
  - Cheap
  - Stable and simple like AdamW

Preconditioner Estimation

- **Idea:** Use **Hessian Diagonal** as Preconditioner
- Approximates curvature per parameter dimension
- Scales updates as:

$$\Delta\theta_i \propto \frac{1}{H_{ii}}$$

- Large curvature → smaller update
- Small curvature → larger update

- **However, computing the Hessian directly is too expensive, so we use an approximation.**

Preconditioner Estimation: Efficient Estimators for Hessian Diagonal

北京大学 PEKING UNIVERSITY

- **Option 1: Hutchinson Estimator**
  - Uses random vectors v to approximate diag(H)
- **Option 2 : Gauss–Newton–Bartlett Estimator**
  - Uses gradients/Jacobians instead of Hessian

- Update **every k steps**, not every step.
- Maintain **EMA** smoothing for stability.

How to estimate Hessian Diagonal?

- **Option 1:** Hutchinson Estimator

$$\hat{h} = u \odot (\nabla^2 \ell(\theta) u)$$

$$\mathbb{E}[\hat{h}] = \mathrm{diag}(\nabla^2 \ell(\theta))$$

Algorithm 1 Hutchinson(θ)

Input: parameter θ.
Compute mini-batch loss $L(\theta)$.
Draw u from $\mathcal{N}(0, \mathrm{I}_d)$.
return $u \odot \nabla(\langle \nabla L(\theta), u \rangle)$.

- Remark: The H-method only needs to compute the Hessian-vector product, which makes it efficient.

How to estimate Hessian Diagonal?

- **Option 2: Gauss–Newton–Bartlett Estimator**
  - The core idea of GNB is to estimate the Hessian diagonal (or, more precisely, the diagonal of its Fisher information) using the **element-wise squared gradient.**

Algorithm 2 Gauss-Newton-Bartlett(θ)

1: **Input:** parameter θ.
2: Draw a mini-batch of input $\{x_b\}_{b=1}^B$.
3: Compute logits on the mini-batch: $\{f(\theta, x_b)\}_{b=1}^B$.
4: Sample $\hat{y}_b \sim \text{softmax}(f(\theta, x_b)), \forall b \in [B]$.
5: Calculate $\hat{g} = \nabla(1/B \sum \ell(f(\theta, x_b), \hat{y}_b))$.
6: **return** $B \cdot \hat{g} \odot \hat{g}$.

GNB: Why Can the Squared Gradient Represent 2nd-Order Curvature?

- **We consider the log-likelihood:**

$$\log p_\theta(y \mid x)$$

- **Negative log-likelihood (NLL):**

$$\ell(\theta; x, y) = -\log p_\theta(y \mid x)$$

- **Define the score (the gradient of the log-likelihood) as:**

$$s(\theta) = \nabla_\theta \log p_\theta(y \mid x).$$

- **Then the gradient of the NLL is:**

$$\nabla_\theta \ell(\theta; x, y) = -s(\theta).$$

GNB: Why Can the Squared Gradient Represent 2nd-Order Curvature?

- **Bartlett Identity:**
  - Under mild regularity conditions, the Bartlett identity states that, for data drawn from the model's own distribution (i.e., $y \sim p_\theta(\cdot \mid x)$),

$$\mathbb{E}_{y\sim p_\theta}[s(\theta)] = 0, \qquad \mathbb{E}_{y\sim p_\theta}\left[s(\theta)s(\theta)^\top\right] = -\mathbb{E}_{y\sim p_\theta}\left[\nabla_\theta^2 \log p_\theta(y \mid x)\right].$$

- **Rewriting in Terms of the NLL:**

$$\mathbb{E}_{y\sim p_\theta}\left[\nabla_\theta \ell \, \nabla_\theta \ell^\top\right] = \mathbb{E}_{y\sim p_\theta}\left[\nabla_\theta^2 \ell\right].$$

- **Interpretation:** This means when labels are sampled from the model distribution, the expected Hessian equals the expected gradient outer product.

GNB: Why Can the Squared Gradient Represent 2nd-Order Curvature?

➢ **Therefore, a very natural estimator for the Hessian diagonal is:**

$$\operatorname{diag}\left(\nabla_\theta^2 \ell\right) \approx \mathbb{E}[(\nabla_\theta \ell) \odot (\nabla_\theta \ell)],$$

where $\odot$ denotes element-wise multiplication.

Bartlett Identity: Mathematical Derivation

We consider a conditional probabilistic model $p_\theta(y \mid x)$, where x is treated as fixed and $\theta \in R^d$ denotes the model parameters. Define the *score function* as

$$s(\theta; y, x) := \nabla_\theta \log p_\theta(y \mid x).$$

All expectations below are taken with respect to $y \sim p_\theta(\cdot \mid x)$.

Regularity assumptions. We assume that:

- $p_\theta(y \mid x)$ is sufficiently smooth in θ;
- differentiation and integration (or summation) can be interchanged;
- the support of $p_\theta(y \mid x)$ does not depend on θ.

Bartlett Identity: Mathematical Derivation

$\boxed{\textbf{First identity: } E[s(\theta)] = 0.}$ Since $p_\theta(y \mid x)$ is a conditional probability distribution, it satisfies

$$\int p_\theta(y \mid x)\, dy = 1.$$

Taking the gradient with respect to θ yields

$$\nabla_\theta \int p_\theta(y \mid x)\, dy = \nabla_\theta 1 = 0.$$

Under the regularity assumptions, we may interchange differentiation and integration:

$$\int \nabla_\theta p_\theta(y \mid x)\, dy = 0.$$

Bartlett Identity: Mathematical Derivation

北京大学 PEKING UNIVERSITY

Using the identity

$$\nabla_\theta p_\theta(y \mid x) = p_\theta(y \mid x)\, \nabla_\theta \log p_\theta(y \mid x),$$

we obtain

$$\int p_\theta(y \mid x)\, s(\theta; y, x)\, dy = 0,$$

which implies

$$\boxed{E_{y \sim p_\theta}\big[s(\theta; y, x)\big] = 0.}$$

Bartlett Identity: Mathematical Derivation

Second identity: $E[s(\theta)s(\theta)^\top] = -E[\nabla_\theta^2 \log p_\theta]$. We start from the second derivative of the log-likelihood:

$$\nabla_\theta \log p_\theta(y \mid x) = \frac{\nabla_\theta p_\theta(y \mid x)}{p_\theta(y \mid x)}.$$

Taking another derivative with respect to θ gives

$$\nabla_\theta^2 \log p_\theta(y \mid x) = \frac{\nabla_\theta^2 p_\theta(y \mid x)}{p_\theta(y \mid x)} - \frac{\nabla_\theta p_\theta(y \mid x)\, \nabla_\theta p_\theta(y \mid x)^\top}{p_\theta(y \mid x)^2}.$$

Noting that

$$\frac{\nabla_\theta p_\theta(y \mid x)}{p_\theta(y \mid x)} = \nabla_\theta \log p_\theta(y \mid x) = s(\theta; y, x),$$

we can rewrite the above as

$$\nabla_\theta^2 \log p_\theta(y \mid x) = \frac{\nabla_\theta^2 p_\theta(y \mid x)}{p_\theta(y \mid x)} - s(\theta; y, x)s(\theta; y, x)^\top.$$

Bartlett Identity: Mathematical Derivation

Multiplying both sides by $p_\theta(y \mid x)$ and integrating over y, we obtain

$$\int p_\theta(y \mid x)\, \nabla_\theta^2 \log p_\theta(y \mid x)\, dy = \int \nabla_\theta^2 p_\theta(y \mid x)\, dy - \int p_\theta(y \mid x)\, s(\theta; y, x) s(\theta; y, x)^\top\, dy.$$

The first term on the right-hand side satisfies

$$\int \nabla_\theta^2 p_\theta(y \mid x)\, dy = \nabla_\theta^2 \int p_\theta(y \mid x)\, dy = \nabla_\theta^2 1 = 0.$$

Therefore,

$$E_{y \sim p_\theta}\left[\nabla_\theta^2 \log p_\theta(y \mid x)\right] = -E_{y \sim p_\theta}\left[s(\theta; y, x) s(\theta; y, x)^\top\right],$$

or equivalently,

$$\boxed{E_{y \sim p_\theta}\left[s(\theta; y, x) s(\theta; y, x)^\top\right] = -E_{y \sim p_\theta}\left[\nabla_\theta^2 \log p_\theta(y \mid x)\right].}$$

Bartlett Identity: Mathematical Derivation

Negative log-likelihood form. Let $\ell(\theta; y, x) = -\log p_\theta(y \mid x)$. Then

$$\nabla_\theta \ell = -s(\theta), \qquad \nabla_\theta^2 \ell = -\nabla_\theta^2 \log p_\theta.$$

Hence, the Bartlett identity can be written as

$$\boxed{E\left[\nabla_\theta \ell \, \nabla_\theta \ell^\top\right] = E\left[\nabla_\theta^2 \ell\right],}$$

which underlies the Gauss–Newton–Bartlett estimator used in second-order optimization methods.

Sophia Update Rule

Algorithm 3 Sophia

Input: θ_1, learning rate $\{\eta_t\}_{t=1}^T$, hyperparameters $\lambda, \gamma, \beta_1, \beta_2, \epsilon$, and estimator choice Estimator $\in$ {Hutchinson, Gauss-Newton-Bartlett}

Set $m_0 = 0$, $v_0 = 0$, $h_{1-k} = 0$

for $t = 1$ **to** T **do**

  Compute minibach loss $L_t(\theta_t)$.

  Compute $g_t = \nabla L_t(\theta_t)$.

  $\boxed{m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t}$

  if $t \bmod k = 1$ **then**

    Compute $\hat{h}_t = \text{Estimator}(\theta_t)$.

    $\boxed{h_t = \beta_2 h_{t-k} + (1 - \beta_2)\hat{h}_t}$

  else

    $h_t = h_{t-1}$

  $\theta_t = \theta_t - \eta_t \lambda \theta_t$ (weight decay)

  $\theta_{t+1} = \theta_t - \eta_t \cdot \text{clip}(m_t / \max\{\gamma \cdot h_t, \epsilon\}, 1)$

Adam-like Momentum Term

Sophia Update Rule

- **Core Update Formula:**

$$\theta_{t+1} = \theta_t - \eta_t \cdot \mathrm{clip}\left(\frac{m_t}{\max(\gamma h_t, \epsilon)}, 1\right)$$

- **Notations:**

m_t: EMA of gradients

h_t: EMA of Hessian diagonal estimate

γ: curvature scaling factor

ϵ: numerical stability

- **Interpretation:**
  - Hessian diagonal acts as adaptive step size
  - Each coordinate has its own curvature-aware scaling
  - Update stabilizer: $\max(\gamma h_t, \epsilon)$

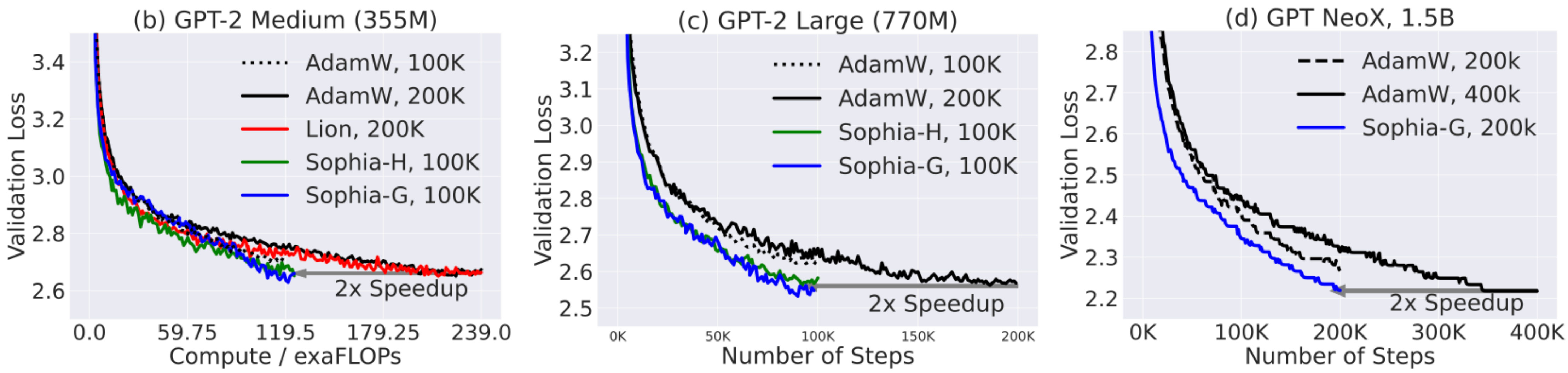
Per-coordinate Clipping

- **Why Needed?**
  - Hessian estimates are noisy
  - High variance may produce excessively large updates
- **Solution: Per-coordinate Clip**
  - Bound each coordinate's update:

$$|u_i| \leq 1$$

  - In practice: ensures stable training even with imperfect curvature

Experiments



(b) GPT-2 Medium (355M). (c) GPT-2 Large (770M). (d) GPT NeoX 1.5B. Across all model sizes, Sophia achieves a 2x speedup.

Summary

- **Sophia is designed to:**
  - Use informative **second-order curvature** (diagonal Hessian)
  - Maintain very **low computation** cost
  - Remain **stable** via per-coordinate clipping
  - Achieve faster convergence than AdamW in large-scale models
- **Key innovation:**
  - A practical second-order optimizer with LLM-scale compatibility.