From Adam to Muon: Effective 'Deep' Optimizer Preconditioners

Yufei Gu

December 21, 2025

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

xLeaf Lab
Machine Learning Foundations

Table of Contents

From Preconditioned gradient methods to Adam

Preconditioning maintains a matrix, termed a preconditioner, to transform the gradient vector before it is used to take a step [3].

- Motivation: In poorly conditioned problems (e.g., long narrow valleys), standard gradient descent zig-zags and converges slowly.
- Goal: Normalize or rescale the space so that optimization is more "isotropic", i.e., behaves similarly in all directions.

In second-order optimization (Newton's method), the update rule is usually:

$$\theta_{t+1} = \theta_t - H^{-1}\nabla_\theta \mathcal{L} \tag{1}$$

- The optimizer preconditioner is the inverse Hessian H^{-1}.
- However, in deep learning, computing H^{-1} is expensive ($O(N^2)/O(N^3)$).
- In convex/quadratic settings, the expected squared gradient correlates with diagonal entries of the Hessian:

$$\mathbb{E}[g_t^2] \approx \text{diag}(H) \tag{2}$$

- Adam preconditioner estimates curvature (H^{-1}) separately per parameter with the diagonal matrix $\frac{1}{\sqrt{\hat{v}_t}+\epsilon}$, avoids off-diagonal estimates [5].

Adam and AdamW: King of Optimizers in DL

AdamW improves Adam with Weight Decay [8]: Better than L2 Regularization (add $\lambda\theta_{t-1}$ to g_t): decouples the optimal choice of weight decay factor from learning rate scheme.

Initial parameters θ_0, learning rate η , weight decay λ, exponential decay rates $\beta_1, \beta_2 \in [0, 1)$, and mini-batch gradient $\bar{g}_t = \sum_{i=0}^{B} g_{t,i}$ at step t.

$$
\begin{aligned}
&m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\bar{g}_t, \quad v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)\bar{g}_t{}^2 \\
&\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t} \\
&\theta_t \leftarrow \theta_{t-1} - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda\theta_{t-1} \right)
\end{aligned}
$$

Adam (AdamW) is considered the King of optimization algorithms (in 2020, and maybe even at present), but if we review the Adam paper:

- The Adam theory was weak; An error was discovered in the proof that Adam will not converge on certain one-dimensional stochastic convex functions.
- The exact same experiments would result in a surefire rejection in these days.
- Knowing that Adam will not always give you the best performance, Adam is considered the default optimizer for deep learning.

Adam is extremely good on deep neural networks and very poor at anything else, e.g., simple convex and non-convex problems [1].

NNs evolved to make Adam the best optimizer

Consider the applied deep learning community:

- The community is like a giant genetic algorithm: researchers are exploring the space of all variants of algorithms and architectures in a semi-random way, consistent success is kept, and the others are discarded.
- Usually, people try new architectures, keeping the optimization algorithm fixed (most of the time Adam).

Hypothesis: People kept evolving new architectures on which Adam works. All the particular choices of architectures in deep learning might have evolved to make Adam work better and better.

Is there a better optimizer for deep learning than Adam/AdamW?

- Efficiency: Less Memory / TFLOPs;
- Performance: Optimization speed and Generalization;
- Consistency: All architecture/tasks in deep learning.

Let's review some potential candidates and their preconditioner design:

- Lion
- Adafactor
- Shampoo
- Muon
- etc.

Symbolic Discovery of Optimization Algorithms [NIPS 2023]

Symbolic Discovery of algorithms (by Google):

1. Defines a train function that takes model weight, gradient, and lr at the current step, and outputs the update to the weight.
2. Include mutations: Insert / Delete / Modify a random statement with randomly chosen functions and arguments.

The search space is infinite and sparse:

- The best program among 2M randomly sampled programs is still significantly inferior to AdamW.
- Search cost is reduced by evolution warm-start and restart, pruning, and low-cost proxies. The final cost is ≈3K TPU V2 days.

And they finally find the 'final' optimizer: Lion.

Lion (Evolved Sign Momentum):

$$
\begin{aligned}
u_t &\leftarrow \text{sign}((1-\beta_1)g_t + \beta_1 m_{t-1}) \\
m_t &\leftarrow (1-\beta_2)g_t + \beta_2 m_{t-1} \\
\theta_t &\leftarrow \theta_{t-1} - \eta(u_t + \lambda\theta_{t-1})
\end{aligned}
$$

- Sign update produces uniform magnitude across all dimensions, and adds noise to the updates as regularization.
- Surpasses Adam (and Adafactor) for a variety of models on different tasks: Transformer, ResNet, U-Net and Hybrid [2].

Evolving with Lion: Leveraging the Sign Function

Experiment Results: [2]

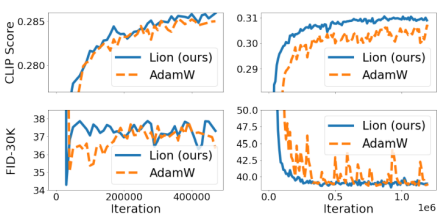


Figure 4: Imagen text-to-image 64^2 (**Left**) and the $64^2 \to 256^2$ diffusion models (**Right**).

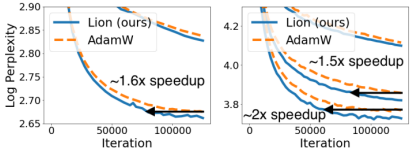


Figure 5: Log perplexity on Wiki-40B (**Left**) and PG-19 (**Right**). The speedup brought by Lion tends to increase with the model scale. The largest model on Wiki-40B is omitted as we observe severe overfitting.

Does Lion come without Cost?

1. Lion does not suit small batch sizes (≤ 64): too much gradient noise.
2. Lion over-optimizes less-frequent token embedding weights:
   - sign(m_t) being a constant;
   - Suggest fix: use AdamW instead.
3. Lion often produces a non-smooth loss curve in training.

If $\beta_1 = \beta_2$ in Lion, we can save more running memory: [11]

Tiger (Tight-first Optimizer):

$$m_t \leftarrow (1-\beta)g_t + \beta m_{t-1}$$
$$\theta_t \leftarrow \theta_{t-1} - \eta\left(\text{sign}(m_t) + \lambda\theta_{t-1}\right)$$

- Performance: Lion $\geq$ Tiger $\geq$ AdamW. (Evaluated only on LLMs.)
- Memory: Tiger $<$ Lion $<$ AdamW.

Cautious Optimizers: Improving Training with One Line of Code [arXiv]

Optimize with Cautious: Do not update unless the update direction is aligned with the current gradient [6].

$$\theta_t \leftarrow \theta_{t-1} - \eta(u_t)$$
$$\Downarrow$$
$$M_t \leftarrow (u_t \times g_t > 0)$$
$$M_t \leftarrow \frac{M_t \times |M_t|}{\sum M_t}$$
$$\theta_t \leftarrow \theta_{t-1} - \eta(u_t \times M_t)$$

- Cautious optimizer will not get stuck at non-stationary points of loss: momentum continues to accumulate and will eventually be updated.
- Cautious optimizer preserves the convergence guarantees of the base optimizer and is compatible to all momentum-based methods.

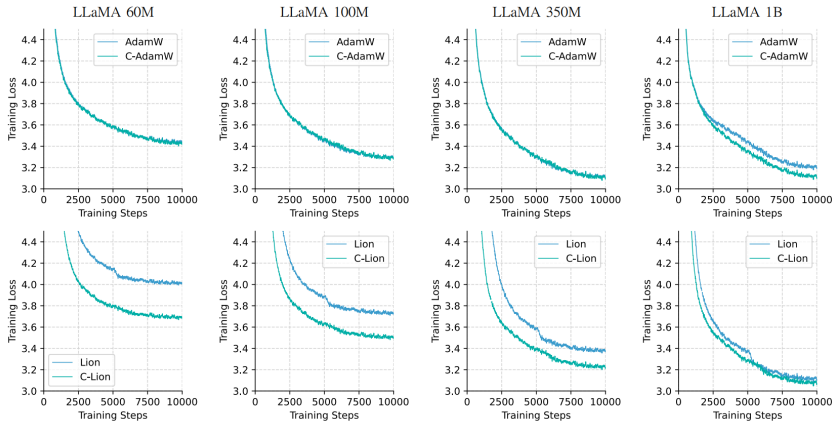


Figure 5: Training loss curves for AdamW, C-AdamW, Lion, C-Lion on LLaMA with 60M, 100M, 350M, and 1B parameters.

Adafactor: Adaptive learning rates with sublinear memory cost [ICML 2018]

Maintaining per-parameter second-moment estimators requires memory equal to the number of parameters.

When storing the full momentum is infeasible, one choice for rank-1 approximation is to have $V = (V1_m)(\frac{1_n^T V}{1_n^T V 1_m})$ which is optimal w.r.t generalized KL divergence:

- $V1_m$ and $1_n^T V$ are the per-row and per-column sums of V.
- By maintaining only the moving average of per-row and per-column sums of squared gradients, the per-parameter second moments can be estimated [10].
- **Adafactor:** preconditioner requires only sublinear cost:

$$
\begin{aligned}
R_t &\leftarrow \beta_2 R_{t-1} + (1-\beta_2) g_t^2 1_m \\
C_t &\leftarrow \beta_2 C_{t-1} + (1-\beta_2) 1_n^T g_t^2 \\
\hat{V}_t &\leftarrow \frac{R_t C_t}{1_n^T R_t} \\
\theta_t &\leftarrow \theta_{t-1} - \eta_t \frac{g_t}{\sqrt{\hat{V}_t} + \epsilon}
\end{aligned}
$$

Adafactor is a rank-1 approximation to the second-order momentum;

What if we use a full-rank approximation instead?

Shampoo: Preconditioned stochastic tensor optimization [ICML 2018]

Shampoo maintains a separate preconditioning matrix H_t^i corresponding to for each dimension $i \in [k]$ of the gradient [3].

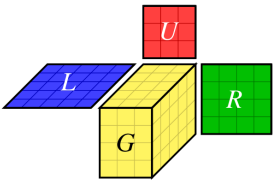


Figure 1. Illustration of Shampoo for a 3-dim tensor $G \in \mathbb{R}^{3\times4\times5}$.

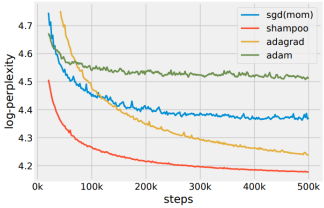


Figure 4. Convergence of test loss for the Transformer model for machine translation (Vaswani et al., 2017) on LM1B.

- The set of preconditioners is updated in an online fashion with the second-order statistics of the accumulated gradients.
- Adaptation to the high-order tensor case is non-trivial, e.g., 4d tensors in CNNs.
- Shampoo, general tensor case ($\times_i$ is the tensor-matrix product):

$$\begin{aligned} &\text{for } i = 1, \ldots, k \text{ do:} \\ &\quad H_t^i \leftarrow H_{t-1}^i + g_t^{(i)} \\ &\theta_t \leftarrow \theta_{t-1} - \eta \cdot g_t \times_i (H_t^i)^{(-1/2k)}, \end{aligned}$$

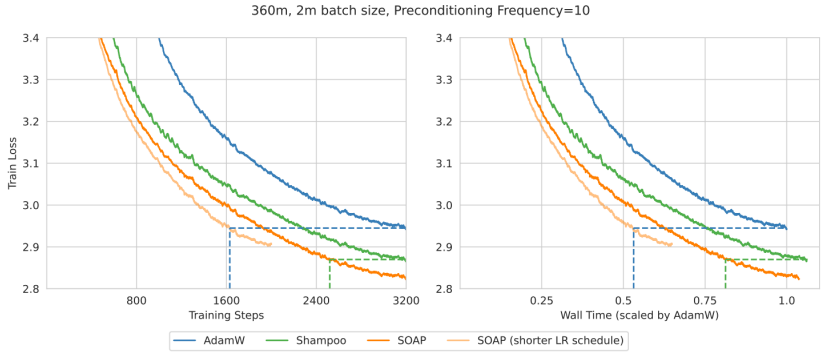
- Space Complexity: $O(\sum_{i=1}^k d_i^2)$.
- Compute Complexity: $O(\sum_{i=1}^k d_i^3)$.
- Step-size decay rate: $O(1/\sqrt{k})$.
- Overall regret bound: $O(\sqrt{T})$.

The gradient outer product approximates the Hessian: Derivation from Fisher information matrix under certain regularity conditions.

$$\mathbb{E}[g_t g_t^T] = \mathbb{E}[\nabla \log p \nabla \log p^T] \approx \mathbb{E}[\nabla_\theta^2 \log p] = \mathbb{E}[H_t(\mathcal{L})]$$

Stepping from Shampoo to Muon

SOAP further improves optimization efficiency by running AdamW in the eigenbasis provided by Shampoo and computes eigendecomposition less frequently [12].



What if we remove preconditioner accumulation in Shampoo?

$$\begin{aligned}\theta_t &\leftarrow \theta_{t-1} - \eta(g_t g_t^T)^{-1/4} g_t (g_t^T g_t)^{-1/4} \\ &= \theta_{t-1} - \eta(US^2U^T)^{-1/4}(USV^T)(VS^2V^T)^{-1/4} \\ &= \theta_{t-1} - \eta(US^{-1/2}U^T)(USV^T)(VS^{-1/2}V^T) \\ &= \theta_{t-1} - \eta US^{-1/2}SS^{-1/2}V^T) \\ &= \theta_{t-1} - \eta UV^T\end{aligned}$$

Given matrix-sign function $\text{sign}_M(A) = \text{sign}_M(U_A \Sigma_A V_A^T) = U_A V_A^T$ ($\Sigma_A^* = I$), we have

$$\theta_t \leftarrow -\eta \underset{M}{\text{sign}}(g_t)$$

We have derived the Muon Optimizer (with no momentum)!

Muon: MomentUm Orthogonalized by Newton-Schulz

Given the Muon intuition derived from Shampoo with sgd-momentum:

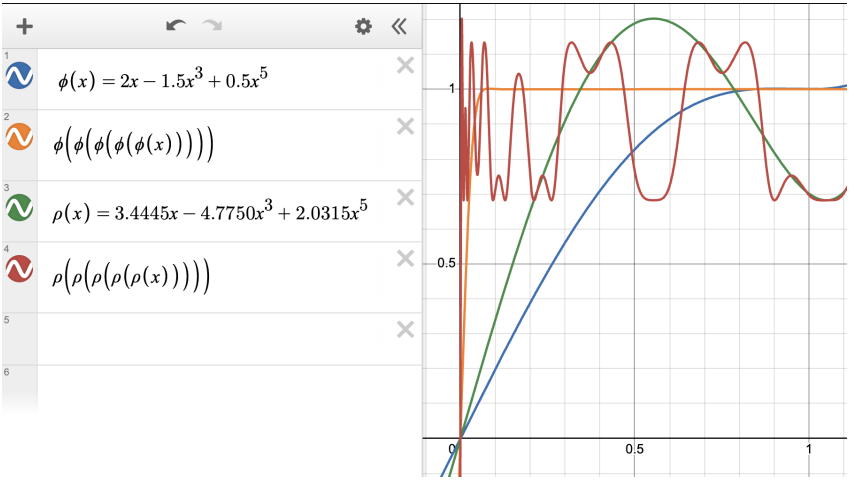
$$m_t \leftarrow \beta m_{t-1} + g_t$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \underset{M}{\text{sign}}(m_t) = \theta_{t-1} - \eta(U_{m_t} V_{m_t}^T)$$

However, performing SVD decomposition on the momentum is costly.

- How to effectively approximate an orthogonal matrix with eigenvalues close to 1, such that $\Sigma_{m_t}^* = I$?

Newton-Schulz iteration: [4]

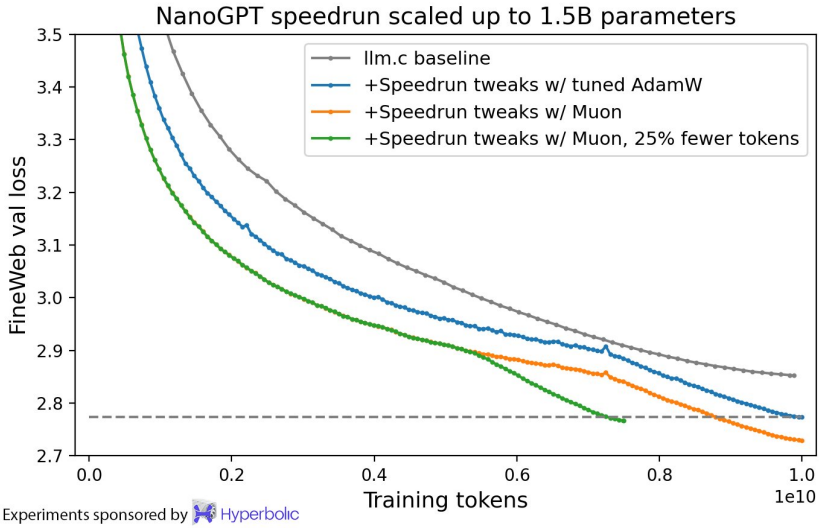
$$X_{t+1} = aX_t + bX_t(X_t^T X_t) + cX_t(X_t^T X_t)^2$$
$$\Rightarrow a = 3.4445, b = 4.7750, c = 2.0315$$



Constant (a, b, c) are optimized/searched toward 5 NS iteration steps.

Muon: MomentUm Orthogonalized by Newton-Schulz

Orthogonalization mat effectively increases the scale of other "rare directions" which have small magnitude in the update but are nevertheless important for learning [4].

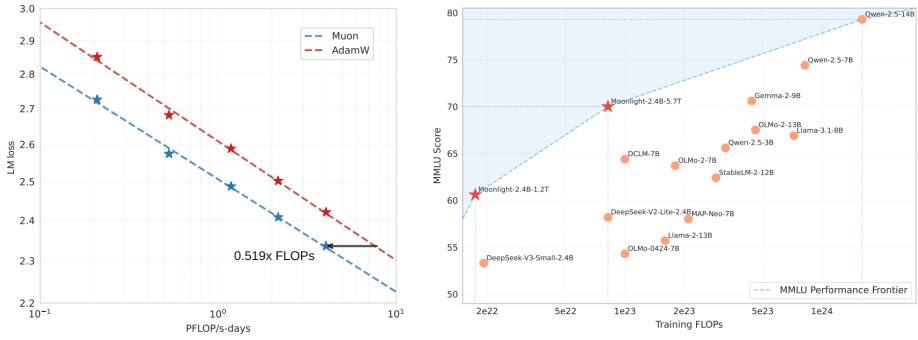


Empirical considerations:

- Muon only applies to 2D parameters; remaining scalar and vector parameters must be optimized using, e.g., AdamW.
- Empirically, input and output parameters optimized using AdamW attained best performance (e.g., embedding and lm_head).

With so many optimizers claimed to beat AdamW, now 'dead', can Muon be really adopted by the deep learning community this time?

Muon is Scalable for LLM Training [Moonshot AI]



(a) (b)

Figure 1: Scaling up with Muon. **(a)** Scaling law experiments comparing Muon and Adam. Muon is $\sim 2\times$ more computational efficient than Adam with compute optimal training. **(b)** The MMLU performance of our Moonlight model optimized with Muon and other comparable models. Moonlight advances the Pareto frontier of performance vs training FLOPs.

Moonshot AI and Essential AI have declared to embrace Muon in 2025:

- Performance gains vs. training FLOPs on 3B/16B-MoE, 5.7T tokens (Moonshot AI) and 3.7B, 160B tokens (Essential AI) [7, 9].

Why focus on Muon?

1. Simplest derivation/implementation among second-order methods.
2. Lightest memory footprint by only maintaining the first momentum.
3. Moonshot AI proposed to rescale Muon's update RMS to align with AdamW, so AdamW hyperparameters can be reused:

$$
\begin{aligned}
m_t &\leftarrow \beta m_{t-1} + g_t \\
O_t &\leftarrow \textbf{Newton-Schulz}(m_t) \\
\theta_t &\leftarrow \theta_{t-1} - \eta(0.2\sqrt{\max(A,B)})O_t + \lambda\theta_{t-1})
\end{aligned}
$$

References

[1] Bremen. Neural networks (maybe) evolved to make adam the best optimizer, 2020.

[2] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023.

[3] V. Gupta, T. Koren, and Y. Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR, 2018.

[4] K. Jordan, Y. Jin, V. Boza, Y. Jiacheng, F. Cesista, L. Newhouse, and J. Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.

[5] D. P. Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] K. Liang, L. Chen, B. Liu, and Q. Liu. Cautious optimizers: Improving training with one line of code. *arXiv preprint arXiv:2411.16085*, 2024.

[7] J. Liu, J. Su, X. Yao, Z. Jiang, G. Lai, Y. Du, Y. Qin, W. Xu, E. Lu, J. Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

[8] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[9] I. Shah, A. M. Polloreno, K. Stratos, P. Monk, A. Chaluvaraju, A. Hojel, A. Ma, A. Thomas, A. Tanwer, D. J. Shah, et al. Practical efficiency of muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

[10] N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[11] J. Shu. Tiger: A tight-first optimizer, Mar 2023.

[12] N. Vyas, D. Morwani, R. Zhao, M. Kwun, I. Shapira, D. Brandfonbrener, L. Janson, and S. Kakade. Soap: Improving and stabilizing shampoo using adam. *arXiv preprint arXiv:2409.11321*, 2024.