Polar Gradient Methods: A Class of Matrix-Gradient Optimizers from a Unifying Preconditioning Perspective

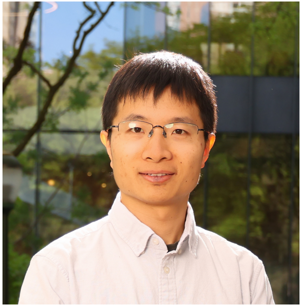
arXiv:2505.21799

February 7, 2026

Tim Tsz-Kit Lau
Department of Statistics and Data Science, The Wharton School
Department of Biostatistics, Epidemiology & Informatics, Perelman School of Medicine
University of Pennsylvania

Collaborators

Prof. Qi Long



Prof. Weijie Su

Departments of Statistics & Data Science and Biostatistics, Epidemiology & Informatics
The Wharton School and Perelman School of Medicine
University of Pennsylvania

The Role of Optimizers Towards AGI

- Optimization methods are the cornerstone for the success of modern large-scale AI (Bottou et al., 2018)
- Pre-training of SOTA base models costs $\gg$ hundreds of millions USD
- Scaling bottlenecks may stem from limitations in
  - Data? ("fossil fuel"" of AI, yet there is only one internet)
  - GPU compute? (Moore's law is no longer valid?)
  - Architecture? (Transformer and attention mechanism)

The Role of Optimizers Towards AGI

- Optimization methods are the cornerstone for the success of modern large-scale AI (Bottou et al., 2018)
- Pre-training of SOTA base models costs $\gg$ hundreds of millions USD
- Scaling bottlenecks may stem from limitations in
  - Data? (“fossil fuel”” of AI, yet there is only one internet)
  - GPU compute? (Moore’s law is no longer valid?)
  - Architecture? (Transformer and attention mechanism)

What about optimizers (training algorithms)?

Which Optimizer Do You Use for Deep Learning?

Adam(W) is the default one

```
model = NeuralNet() # torch.nn.Module
params = model.parameters()
```

```
optimizer = torch.optim.AdamW(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.01)
```

Which Optimizer Do You Use for Deep Learning?

Adam(W) is the default one

```
model = NeuralNet() # torch.nn.Module
params = model.parameters()

optimizer = torch.optim.AdamW(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.01)
```

Update rules of Adam(W) (Kingma and Ba, 2015; Loshchilov and Hutter, 2019):

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k, \qquad g_k = \nabla f(w_k)$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^{\odot 2}$$

$$\widehat{m}_k = \frac{m_k}{1-\beta_1^k}$$

$$\widehat{v}_k = \frac{v_k}{1-\beta_2^k}$$

$$w_{k+1} = (1-\lambda\gamma_k) w_k - \gamma_k \frac{\widehat{m}_k}{\sqrt{\widehat{v}_k} + \varepsilon}$$

The Impact of Adam

Adam: A method for stochastic optimization
DP Kingma, J Ba
arXiv preprint arXiv:1412.6980

239553 2014

(retrieved Jan 19, 2026)

ICLR
International Conference On Learning Representations

Announcing the Test of Time Award Winners from ICLR 2015

CARL VONDRICK / ICLR 2025

We are honored to announce the Test of Time awards for ICLR 2025. This award recognizes papers published ten years ago at ICLR 2015 that have had a lasting impact on the field. The 2025 program chairs and general chair reviewed the papers published at ICLR 2015, and selected the two papers below for their profound influence and impact on machine learning today.

Congratulations to the authors of the Test of Time winner and runner up!

Test of Time

Adam: A Method for Stochastic Optimization
Diederik P. Kingma, Jimmy Ba
https://arxiv.org/abs/1412.6980

Adam as a Vector Preconditioned Gradient Method

- Adam's update rules are:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k \qquad \widehat{m}_k = m_k/(1-\beta_1^k)$$
$$v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^{\odot 2} \qquad \widehat{v}_k = v_k/(1-\beta_2^k)$$
$$w_{k+1} = w_k - \gamma_k \widehat{m}_k / \left(\sqrt{\widehat{v}_k} + \varepsilon\right)$$

- Define a diagonal preconditioner $P_k = \mathrm{Diag}\left(\sqrt{\widehat{v}_k} + \varepsilon\right)$ and denote $\|w\|_P := \sqrt{\langle w, Pw\rangle}$

$$w_{k+1} = \operatorname*{argmin}_{w \in \mathbb{R}^d} \left\{ \langle \widehat{m}_k, w - w_k \rangle + \frac{1}{2\gamma_k} \|w - w_k\|_{P_k}^2 \right\} = w_k - \gamma_k P_k^{-1} \widehat{m}_k$$

- Both $\widehat{m}_k$ and P_k are functions of g_k; in Gauss–Newton, $\nabla^2 f(w_k) \approx g_k g_k^\top$
- Majorization-minimization methods? Unclear if $\nabla^2 f(w_k) \preccurlyeq P_k$ always holds (unlikely here; P_k is just a Hessian approximation)

Adam as (Smoothed) *Normalized* Steepest Descent w.r.t. ℓ_∞-Norm

- signSGD (i.e., Adam with $\beta_1 = \beta_2 = 0$; Bernstein et al., 2018):

$$w_{k+1} = w_k - \gamma_k \cdot \mathrm{sgn}(g_k),$$

  i.e., normalized steepest descent w.r.t. (squared) ℓ_∞-norm: $\|w\|_\infty = \max_{1 \leqslant i \leqslant d} |w_i|$

- Unnormalized steepest descent w.r.t. (squared) ℓ_∞-norm (with dual norm scaling):

$$\begin{aligned} w_{k+1} &= \operatorname*{argmin}_{w \in \mathbb{R}^d} \left\{ \langle g_k, w - w_k \rangle + \frac{1}{2\gamma_k} \|w - w_k\|_\infty^2 \right\} \\ &= w_k - \gamma_k \cdot \|g_k\|_1 \mathrm{sgn}(g_k) \end{aligned}$$

Parameters are Matrices instead of Vectors

Multilayer perceptron for classification:

$$\mathbb{P}(x; W) = \mathrm{softmax}(W_\ell \sigma(W_{\ell-1}\sigma(\cdots\sigma(W_1 x)\cdots)))$$

- $\sigma(\cdot)$ nonlinear activation; e.g., $\sigma(x) = \mathrm{ReLU}(x) = \max\{x, 0\}$; bias omitted
- The size of W_i is

  #neurons in previous layer $\times$ #neurons in next layer

- Parameters are predominately matrices in all architectures (e.g., fully connected layers; QKV in transformer)

Empirical Evidence: The AlgoPerf (Training Algorithms) Competition

- The AlgoPerf: Training Algorithms competition (Dahl et al., 2023; Kasimbeg et al., 2025) aims at evaluating practical speed-ups in neural network training achieved *solely by improving the underlying training algorithm*
- Winner: **Distributed Shampoo** (Shi et al., 2023), an optimizer that does not treat parameters and gradients as vectors

Empirical Evidence: The AlgoPerf (Training Algorithms) Competition

- The AlgoPerf: Training Algorithms competition (Dahl et al., 2023; Kasimbeg et al., 2025) aims at evaluating practical speed-ups in neural network training achieved *solely by improving the underlying training algorithm*
- Winner: **Distributed Shampoo** (Shi et al., 2023), an optimizer that does not treat parameters and gradients as vectors

Takeaway

Results challenge belief in Adam's optimality for deep learning

A New Optimizer in 2024

Keller Jordan blog

Muon: An optimizer for hidden layers in neural networks

December 8, 2024 · 17 min

Muon is an optimizer for the hidden layers in neural networks. It is used in the current training speed records for both NanoGPT and CIFAR-10 speedrunning.

- Blog post popularized on X, not published
- Leading author Keller Jordan joined OpenAI

Algorithm 2 Muon

Require: Learning rate η, momentum μ

Initialize $B_0 \leftarrow 0$
for $t = 1, \ldots$ **do**
    Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
    $B_t \leftarrow \mu B_{t-1} + G_t$
    $O_t \leftarrow \text{NewtonSchulz5}(B_t)$
    Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta O_t$
end for
return θ_t

The Muon Optimizer (Jordan et al., 2024)

Muon

- Let $G_k = \nabla f(W_k) = U_k \Sigma_k V_k^\top \in \mathbb{R}^{m \times n}$ be the SVD
- The new update:

$$W_{k+1} = W_k - \gamma_k U_k V_k^\top$$

The Muon Optimizer (Jordan et al., 2024)

Muon

- Let $G_k = \nabla f(W_k) = U_k \Sigma_k V_k^\top \in \mathbb{R}^{m \times n}$ be the SVD
- The new update:

$$W_{k+1} = W_k - \gamma_k U_k V_k^\top$$

- In practice, momentum is used: $M_k = \beta M_{k-1} + (1 - \beta) G_k$
- Earlier ideas: Carlson et al. (2016); Gupta et al. (2018); Vyas et al. (2025)

The Muon Optimizer (Jordan et al., 2024)

Muon

- Let $G_k = \nabla f(W_k) = U_k \Sigma_k V_k^\top \in \mathbb{R}^{m \times n}$ be the SVD
- The new update:

$$W_{k+1} = W_k - \gamma_k U_k V_k^\top$$

- In practice, momentum is used: $M_k = \beta M_{k-1} + (1-\beta) G_k$
- Earlier ideas: Carlson et al. (2016); Gupta et al. (2018); Vyas et al. (2025)

- $UV^\top$ is the projection of $\nabla f(W_k)$ onto the semi-orthogonal space $\mathcal{O}^{m \times n} := \{A \in \mathbb{R}^{m \times n} : A^\top A = I_n$ or $AA^\top = I_m\}$
- $\mathrm{msgn}(X) := UV^\top$ extends matrix sign function for (symmetric) square matrices: $X = U \Sigma U^\top$, then $\mathrm{msgn}(X)$ coincides with $U \mathrm{sgn}(\Sigma) U^\top$

Finding $\operatorname{msgn}(X)$ via Newton–Schulz Iteration

To be GPU-friendly, Jordan et al. (2024) used polynomial-based iterations to approximate $\operatorname{msgn}(X)$:

- $\operatorname{msgn}(X) - p(X) = U(I - p(\Sigma))\,V^\top$, for odd polynomial

$$p(X) := a_1 X + a_3 X(X^\top X) + \cdots + a_{2q+1} X(X^\top X)^q$$

- Example: quintic polynomial $p(x) = \frac{15}{8}x - \frac{5}{4}x^3 + \frac{3}{8}x^5$
- Amsel et al. (2025, Polar Express) suggested approximation via compositions of polynomials:

$$p^\star = \operatorname*{argmin}_{p \in \mathcal{P}_d^{odd}} \max_{x \in [\ell, u]} |1 - p(x)|$$

Finding $\mathrm{msgn}(X)$ via Newton–Schulz Iteration

To be GPU-friendly, Jordan et al. (2024) used polynomial-based iterations to approximate $\mathrm{msgn}(X)$:

- $\mathrm{msgn}(X) - p(X) = U(I - p(\Sigma))\,V^\top$, for odd polynomial

$$p(X) := a_1 X + a_3 X(X^\top X) + \cdots + a_{2q+1} X(X^\top X)^q$$

- Example: quintic polynomial $p(x) = \frac{15}{8}x - \frac{5}{4}x^3 + \frac{3}{8}x^5$
- Amsel et al. (2025, Polar Express) suggested approximation via compositions of polynomials:

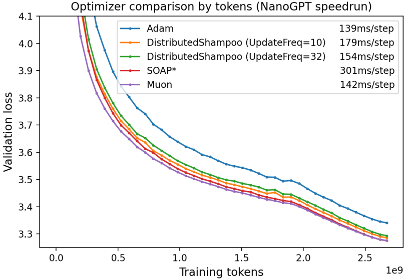
$$p^\star = \operatorname*{argmin}_{p \in \mathcal{P}_d^{odd}} \max_{x \in [\ell, u]} |1 - p(x)|$$

- Weirdly, more accurate (polynomial) approximation doesn't yield better LLM optimization

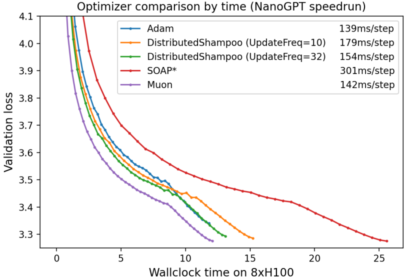
Early Promising Experimental Results (Jordan et al., 2024)

Why it's difficult to beat Adam

Neural architectures (like Transformer) may be "overfitted" to Adam's optimization characteristics (Orabona, 2020)



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

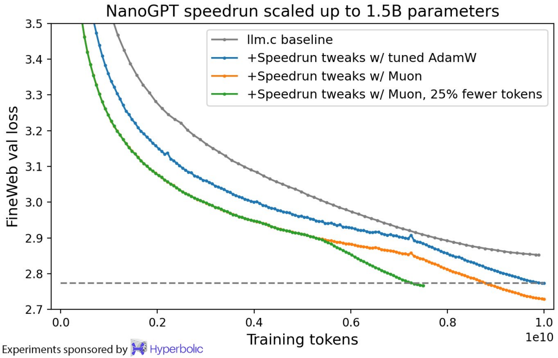


*SOAP is under active development. Future versions will significantly improve the wallclock overhead.

Early Promising Experimental Results (Jordan et al., 2024)

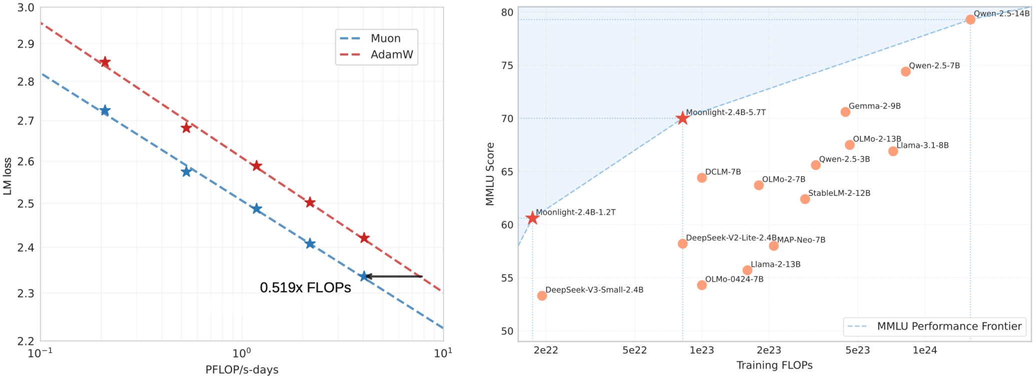
Why it's difficult to beat Adam

Neural architectures (like Transformer) may be "overfitted" to Adam's optimization characteristics (Orabona, 2020)



Superiority of Muon over Adam on 16B LLMs by Moonshot (Liu et al., 2025)

Muon is about $2\times$ more computationally efficient than AdamW



(a) (b)

Questions This Talk Will Address

- Why is **orthogonalization** good for deep learning optimization?
- How can we improve **Muon** from an algorithmic perspective?

A Unifying Preconditioning Perspective on Matrix-Gradient Methods

Curvature Information is (Completely) Missing in Muon

Curvature information from singular values are completely gone

Curvature Information is (Completely) Missing in Muon

Curvature information from singular values are completely gone

- Moonshot's tuning parameters for Muon (Liu et al., 2025):

$$W_{k+1} = W_k - 0.2\gamma_k\sqrt{\max\{m, n\}} \cdot \mathrm{msgn}(G_k)$$

- $\{\gamma_k\}$ are pre-specified
- Oscillate even if $G_k \to 0$

Definition (Null-gradient consistency)

An optimization algorithm exhibits null-gradient consistency if the magnitude of its update step tends to zero as the effective gradient term approaches zero

Curvature (Partially) Recovered via Polar Decomposition

- Polar decomposition: $U_{\mathsf{p}} H = \mathrm{polar}(X)$
  - $U_{\mathsf{p}} \in \mathbb{O}^{m \times n}$ has orthonormal columns
  - $H \in \mathbb{S}_{+}^{n}$ is a symmetric positive semidefinite matrix
- If $U \Sigma V^{\top} = \mathrm{SVD}(X)$, then $U_{\mathsf{p}} = U V^{\top} = \mathrm{msgn}(X)$ and $H = V \Sigma V^{\top}$
- H contains the curvature information

PolarGrad

- Partial curvature information can then be recovered since the nuclear norm is the sum of singular values
- Matrix sign descent with polar decomposition of gradient:

$$U\Sigma V^\top = \mathrm{SVD}(G) \Rightarrow \|G\|_{\mathrm{nuc}} = \mathrm{tr}(\Sigma)$$

$$\mathrm{tr}(H) = \mathrm{tr}(V\Sigma V^\top) = \mathrm{tr}(V^\top V\Sigma) = \mathrm{tr}(\Sigma)$$

PolarGrad

$$U_k H_k = \mathrm{polar}(G_k), \quad W_{k+1} = W_k - \gamma_k \,\mathrm{tr}(H_k)\, U_k$$

- Step size/learning rate matters!

It's Muon with Armijo's Backtracking Line Search

- Determine a learning rate $\alpha_k > 0$ such that:
  $$f(X_k - \alpha_k U_k) \leqslant f(X_k) - c\alpha_k \langle\!\langle G_k, U_k \rangle\!\rangle_{\mathrm{F}} = f(X_k) - c\alpha_k |\!|\!| G_k |\!|\!|_{\mathrm{nuc}}, \ 0 < c < 1$$
- If f is L-Lipschitz smooth , then
  $$f(X_k - \alpha_k U_k) \leqslant f(X_k) - \alpha_k |\!|\!| G_k |\!|\!|_{\mathrm{nuc}} + \frac{L}{2}\alpha_k^2 r_k, \ r_k := \mathrm{rank}(G_k)$$
- Hence, the learning rate satisfies
  $$\alpha_k \leqslant \frac{2(1-c)}{L r_k} |\!|\!| G_k |\!|\!|_{\mathrm{nuc}}$$
- Backtracking line search keeps $\alpha_k / |\!|\!| G_k |\!|\!|_{\mathrm{nuc}}$ in a stable range
- Implicitly recovers the nuclear norm scaling term

It's Muon with Armijo's Backtracking Line Search

- Determine a learning rate $\alpha_k > 0$ such that:
  $$f(X_k - \alpha_k U_k) \leqslant f(X_k) - c\alpha_k \langle\!\langle G_k, U_k \rangle\!\rangle_{\mathrm{F}} = f(X_k) - c\alpha_k |\!|\!| G_k |\!|\!|_{\mathrm{nuc}}, \;\; 0 < c < 1$$
- If f is L-Lipschitz smooth , then
  $$f(X_k - \alpha_k U_k) \leqslant f(X_k) - \alpha_k |\!|\!| G_k |\!|\!|_{\mathrm{nuc}} + \frac{L}{2}\alpha_k^2 r_k, \;\; r_k := \mathrm{rank}(G_k)$$
- Hence, the learning rate satisfies
  $$\alpha_k \leqslant \frac{2(1-c)}{L r_k} |\!|\!| G_k |\!|\!|_{\mathrm{nuc}}$$
- Backtracking line search keeps $\alpha_k / |\!|\!| G_k |\!|\!|_{\mathrm{nuc}}$ in a stable range
- Implicitly recovers the nuclear norm scaling term

Why explicit scaling in PolarGrad?

- Backtracking line search is computationally expensive and rarely used in DL
- PolarGrad makes the scaling explicit: $\gamma_k \, \mathrm{tr}(H_k) \equiv \gamma_k |\!|\!| G_k |\!|\!|_{\mathrm{nuc}}$

PolarGrad as Spectral-Norm-Regularized Steepest Descent

$$U_k H_k = \mathrm{polar}(G_k), \qquad W_{k+1} = W_k - \gamma_k \, \mathrm{tr}(H_k) \, U_k$$

$$W_k - \gamma_k \, \mathrm{tr}(H_k) \, U_k = \operatorname*{argmin}_{W \in \mathbb{R}^{m \times n}} \left\{ \langle\!\langle G_k, W - W_k \rangle\!\rangle_{\mathrm{F}} + \frac{1}{2\gamma_k} |\!|\!| W - W_k |\!|\!|_{\mathrm{S}}^2 \right\}$$

- Satisfies the null-gradient consistency
- Spectral norm is submultiplicative: $|\!|\!| XY |\!|\!| \leqslant |\!|\!| X |\!|\!| \, |\!|\!| Y |\!|\!|$, and any unitarily invariant matrix norm satisfies $|\!|\!| X |\!|\!| \geqslant |\!|\!| X |\!|\!|_{\mathrm{S}}$

PolarGrad as Spectral-Norm-Regularized Steepest Descent

$$U_k H_k = \operatorname{polar}(G_k), \qquad W_{k+1} = W_k - \gamma_k \operatorname{tr}(H_k)\, U_k$$

$$W_k - \gamma_k \operatorname{tr}(H_k)\, U_k = \operatorname*{argmin}_{W \in \mathbb{R}^{m \times n}} \left\{ \langle\!\langle G_k, W - W_k \rangle\!\rangle_{\mathrm{F}} + \frac{1}{2\gamma_k} |\!|\!| W - W_k |\!|\!|_{\mathrm{S}}^2 \right\}$$

- Satisfies the null-gradient consistency
- Spectral norm is submultiplicative: $|\!|\!| XY |\!|\!| \leqslant |\!|\!| X |\!|\!| \, |\!|\!| Y |\!|\!|$, and any unitarily invariant matrix norm satisfies $|\!|\!| X |\!|\!| \geqslant |\!|\!| X |\!|\!|_{\mathrm{S}}$

However, Adam's ($\beta_1 = \beta_2 = 0$) update rules for matrix parameters is

$$W_{k+1} \in \operatorname*{Argmin}_{W \in \mathbb{R}^{m \times n}} \left\{ \langle\!\langle G_k, W - W_k \rangle\!\rangle_{\mathrm{F}} + \frac{1}{2\gamma_k} |\!|\!| W - W_k |\!|\!|_{\max}^2 \right\},$$

where $|\!|\!| W |\!|\!|_{\max} = \max_{1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n} |w_{i,j}|$ is NOT a submultiplicative norm

Matrix Preconditioned Gradient Methods

$$X_{k+1} = X_k - \gamma_k \mathscr{P}_k(\nabla \mathsf{f}(X_k)),$$

where $\mathscr{P}_k \colon \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ is a *preconditioning function* of the gradient

- Vector preconditioned gradient methods are curvature-aware and aim to reduce the (local) condition number of the Hessian $\kappa_2(\nabla^2 \mathsf{f}(X))$:

Curvature-Anisotropy Preconditioning

- Another preconditioning concept for matrix optimization problems:

Gradient-Anisotropy Preconditioning

- Minimizes the condition number of the matrix-valued gradient at each step

$$\kappa_2(\nabla \mathsf{f}(X)) := \frac{\sigma_{\max}(\nabla \mathsf{f}(X))}{\sigma_{\min}(\nabla \mathsf{f}(X))}$$

- Orthogonal matrices are the "*best conditioned*" ones, with condition numbers of 1

Vector versus Matrix PGMs

- Sign descent or signSGD (Bernstein et al., 2018) (Adam with $\beta_1 = \beta_2 = 0$):

$$w_{k+1} = \operatorname*{argmin}_{w \in \mathbb{R}^d} \left\{ \langle g_k, w - w_k \rangle + \frac{1}{2\gamma_k} \|w - w_k\|_\infty^2 \right\} = w_k - \gamma_k \|g_k\|_1 \cdot \operatorname{sgn}(g_k)$$

- PolarGrad:

$$W_{k+1} \in \operatorname*{Argmin}_{W \in \mathbb{R}^{m \times n}} \left\{ \langle\!\langle G_k, W - W_k \rangle\!\rangle_{\mathrm{F}} + \frac{1}{2\gamma_k} \|\!| W - W_k \|\!|_{\mathrm{S}}^2 \right\} = W_k - \gamma_k \|\!| G_k \|\!|_{\mathrm{nuc}} \cdot \operatorname{msgn}(G_k)$$

- sgn: only gives entrywise sign; preconditioning effect is inconclusive; potential cause for training instabilities
- msgn: sets all singular values to 1; maintains the original update directions given by the singular vectors

Convergence Analysis

Convergence Analysis

Theorem

Suppose that $f : \mathbb{R}^{m\times n} \to \overline{\mathbb{R}}$ *is* L*-Lipschitz smooth and a* μ*-PŁ function, then with* $\gamma_k = 1/(Lr_k)$

$$f(X_{k+1}) - f^\star \leqslant (1 - 1/(r_k\kappa_H))(f(X_k) - f^\star),$$
$$f(X_{k+1}) - f^\star \leqslant \left(1 - 1/(\kappa_{G_k}^2\kappa_H)\right)(f(X_k) - f^\star),$$

where $r_k := \mathrm{rank}(\nabla f(W_k))$, $\kappa_{G_k} := \sigma_1(\nabla f(X_k))/\sigma_{r_k}(\nabla f(X_k))$, $\kappa_H := L/\mu$

- Gradient-based rate can significantly outperform the Hessian-based rate when $\kappa_{G_k} \ll r_k$

Convergence Analysis

Theorem

Suppose that $f: \mathbb{R}^{m\times n} \to \overline{\mathbb{R}}$ *is* L*-Lipschitz smooth and a* μ*-PŁ function, then with* $\gamma_k = 1/(Lr_k)$

$$f(X_{k+1}) - f^\star \leqslant (1 - 1/(r_k\kappa_H))(f(X_k) - f^\star),$$
$$f(X_{k+1}) - f^\star \leqslant \left(1 - 1/(\kappa_{G_k}^2\kappa_H)\right)(f(X_k) - f^\star),$$

where $r_k := \mathrm{rank}(\nabla f(W_k))$, $\kappa_{G_k} := \sigma_1(\nabla f(X_k))/\sigma_{r_k}(\nabla f(X_k))$, $\kappa_H := L/\mu$

- Gradient-based rate can significantly outperform the Hessian-based rate when $\kappa_{G_k} \ll r_k$

Theorem

Assume unbiased stochastic gradients with bounded variance $\varsigma^2 \in (0, \infty)$*, then* $f(W_K) - f^\star \leqslant \mathscr{O}(1/K)$

Inexact Polar Oracles via Numerical Polar Decomposition Algorithms

$\mathrm{msgn}(X)$ is Only Approximated Numerically In Practice

- Recall that Muon relies on Newton–Schulz iteration (for polar decomposition of gradient/momentum)
- In general, we can use any numerical polar decomposition algorithms (inexact polar oracles $\widehat{\mathrm{polar}}$)

$$\widetilde{U}_k \widetilde{H}_k = \widehat{\mathrm{polar}}(G_k), \quad W_{k+1} = W_k - \gamma_k \widetilde{\nu}_k \widetilde{U}_k,$$

where the nuclear norm scaling is computed using $\widetilde{\nu}_k := \left\langle\!\left\langle \widetilde{U}_k, G_k \right\rangle\!\right\rangle_{\mathrm{F}}$

Alternative Numerical Polar Decomposition Algorithms

- The Polar Express (Amsel et al., 2025): polynomial iterations
- QR-based Dynamically Weighted Halley (QDWH) (Nakatsukasa and Higham, 2013): rational iterations
- ZOLO-based Polar Decomposition (ZOLO-PD) (Nakatsukasa and Freund, 2016): a higher-order variant of QDWH

How to Choose Inexact Polar Oracles?

- Both the NS iteration and QDWH give cubic convergence of orthogonality error $e_{j+1} \leqslant \zeta e_j^3$ for some $\zeta > 0$, where $e_j := \| \widetilde{U}_j^\top \widetilde{U}_j - I \|_{\mathrm{S}}$
- ζ_{NS} depends strongly on $e_0 = 1 - 1/\kappa_2(G)^2$ since NS is a polynomial iteration
- If G is so ill-conditioned, ζ_{NS} could be unbounded, the NS iteration loses its cubic convergence behavior and may even diverge without additional rescaling

How to Choose Inexact Polar Oracles?

- Both the NS iteration and QDWH give cubic convergence of orthogonality error $e_{j+1} \leqslant \zeta e_j^3$ for some $\zeta > 0$, where $e_j := \|\!|\widetilde{U}_j^\top \widetilde{U}_j - I\|\!|_{\mathrm{S}}$
- ζ_{NS} depends strongly on $e_0 = 1 - 1/\kappa_2(G)^2$ since NS is a polynomial iteration
- If G is so ill-conditioned, ζ_{NS} could be unbounded, the NS iteration loses its cubic convergence behavior and may even diverge without additional rescaling
- ζ_{QDWH} is bounded and does not blow up as $\kappa_2(G) \to \infty$ because its rational part $(I + c_j M_j)^{-1}$ compresses large singular values and stretches small ones
- QDWH keeps the iteration centered at the optimal cubic fixed point
- QDWH is indeed *provably stable* and *cubically convergent* even when $\kappa_2(G) = 10^{16}$ (Nakatsukasa et al., 2010)

How to Choose Inexact Polar Oracles?

Further assume $\|\| \widetilde{U}_k - U_k \|\|_{\mathrm{S}} \leqslant \varepsilon_k$ for some $0 \leqslant \varepsilon_k < 1$, and $\|\| \widetilde{U}_k^\top \widetilde{U}_k - I \|\|_{\mathrm{S}} = \mathscr{O}(\delta_k)$ for some $\delta_k \geqslant 0$

Theorem (PolarGrad with Inexact Polar Oracles)

Running NS or QDWH for T inner steps so that $\widetilde{U}_k = \widetilde{U}_{k,T}$, we have the oracle error bounds $\varepsilon_{\max}(T) = \mathscr{O}(e_0^{3^T})$ and $\delta_{\max}(T) = \mathscr{O}(e_0^{3^T})$, where $e_{k,j} := \|\| \widetilde{U}_{k,j}^\top \widetilde{U}_{k,j} - I \|\|_{\mathrm{S}}$ for $k \in \{0, \ldots, K\}$ and $j \in \{0, \ldots, T\}$, and $e_0 := \max_{k \in \{0,\ldots,K\}} e_{k,0}$. Therefore, when running realized PolarGrad*, to stay within $1 - \eta$ of the exact rate for some $\eta \in (0,1)$, it requires at least $\lceil \mathscr{O}(\log(\log \eta / \log e_0)) \rceil$ inner steps.*

How to Choose Inexact Polar Oracles?

Factors for consideration:

- Computational cost (NS has lower FLOPS)
- Required numerical precision (QDWH/ZOLO-PD requires high precision)
- Numerical stability (NS is not numerically stable for ill-conditioned gradient)
- Hardware consideration such as GPU-friendliness of involved operations (NS is GPU-friendly)
- The complexity of the operations involved (QR decomposition or matrix inversion)

How to Choose Inexact Polar Oracles?

Factors for consideration:

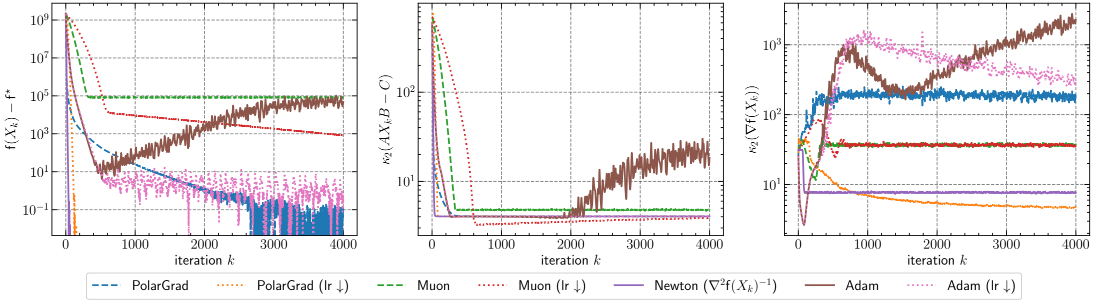
- Computational cost (NS has lower FLOPS)
- Required numerical precision (QDWH/ZOLO-PD requires high precision)
- Numerical stability (NS is not numerically stable for ill-conditioned gradient)
- Hardware consideration such as GPU-friendliness of involved operations (NS is GPU-friendly)
- The complexity of the operations involved (QR decomposition or matrix inversion)

Suggestions

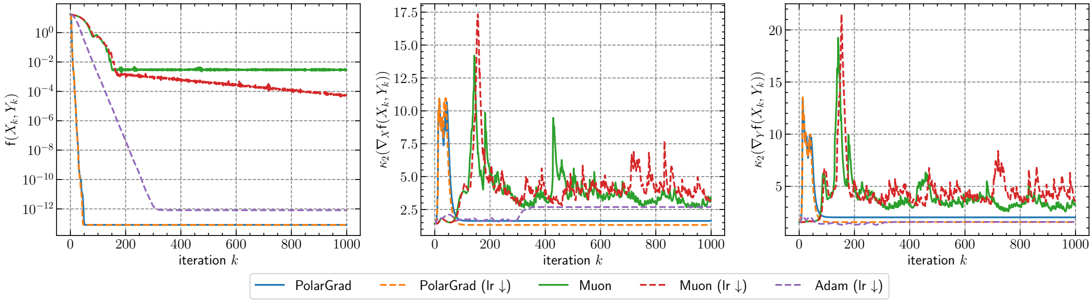
- NS/Polar Express: deep learning (linear and attention layers)
- QDWH:
  - ill-conditioned gradient/momentum matrices
  - smaller-scale matrix optimization problems on CPUs
  - high precision scenarios

Numerical Experiments

Matrix Quadratic Regression $\left(f(X) = \frac{1}{2}\|AXB - C\|_F^2\right)$



Low-Rank Matrix Completion $(f(X, Y) = \| (XY^\top - M_\star)_{\text{obs}} \|_F^2)$



Qwen2.5 Pre-Training

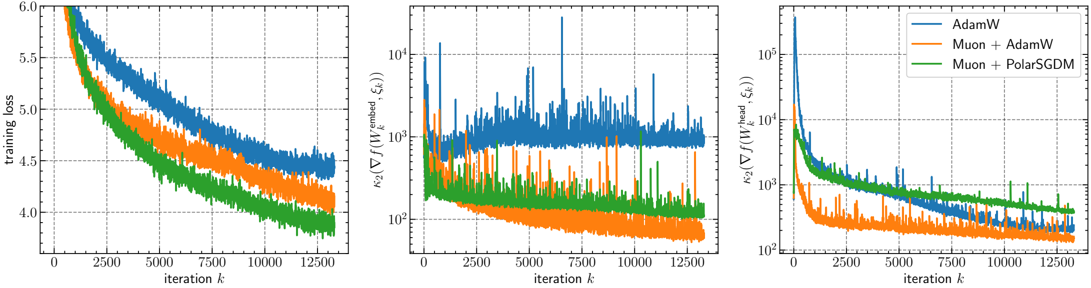


Figure: AdamW—AdamW for all parameters; Muon + AdamW (PolarSGDM)—Muon for hidden layers and AdamW (PolarSGDM) for embedding and head layers

Further Results

Optimizers for Embedding and Head Layers

- Training runs with Muon still use Adam(W) for the input embedding and head layers
- A mismatch from the theoretical choice of norms for steepest descent (Bernstein and Newhouse, 2025)

Optimizers for Embedding and Head Layers

- Training runs with Muon still use Adam(W) for the input embedding and head layers
- A mismatch from the theoretical choice of norms for steepest descent (Bernstein and Newhouse, 2025)
- Input embedding matrix $E \in \mathbb{R}^{V \times d}$, where V is the vocabulary size and d is the embedding dimension with $V \gg d$

Optimizers for Embedding and Head Layers

- Training runs with Muon still use Adam(W) for the input embedding and head layers
- A mismatch from the theoretical choice of norms for steepest descent (Bernstein and Newhouse, 2025)
- Input embedding matrix $E \in \mathbb{R}^{V \times d}$, where V is the vocabulary size and d is the embedding dimension with $V \gg d$
- Input embedding's gradient $G_E = S^\top M$, where $S \in \mathbb{R}^{b \times V}$ is a sparse token-selection matrix (one-hot), $M \in \mathbb{R}^{b \times d}$ is a dense backpropagated signal and b is the batch size
- G_E is rank-deficient since $\mathrm{rank}(G) \leqslant \min\{b, d\} \ll d$ and fluctuates with batch composition $\Rightarrow \kappa_2(G_E) \to 0$
- For stochastic gradient, the small singular values are thus dominated by stochastic noise
- Thus, for the input embedding, the NS iteration would diverge

Optimizers for Embedding and Head Layers

- Head matrix $W \in \mathbb{R}^{V \times d}$ is even worse than token embeddings
- Head matrix's gradient G_W is driven by softmax logits with highly skewed distributions where rare tokens get near-zero signal $\Rightarrow$ even more ill-conditioned gradient

Optimizers for Embedding and Head Layers

- Head matrix $W \in \mathbb{R}^{V \times d}$ is even worse than token embeddings
- Head matrix's gradient G_W is driven by softmax logits with highly skewed distributions where rare tokens get near-zero signal $\Rightarrow$ even more ill-conditioned gradient
- Even though Adam does not compute a polar direction, it implicitly applies a *diagonal rational preconditioner*
- However, the diagonal structure does not capture correlations across the embedding space and completely ignores the matrix geometry
- QDWH-PolarGrad could be more desired if d is small or moderate, or QDWH is performed infrequently and cheaper updates are kept in between

Connecting signSGD (on Matrices) to PolarGrad

- Recover unnormalized signSGD from PolarGrad by embedding a vector variable as a diagonal matrix
- "Matrize" $w \in \mathbb{R}^d$ as the diagonal matrix $D := \mathrm{Diag}(w) \in \mathbb{R}^{d \times d}$
- Define $F \colon \mathbb{R}^{d \times d} \to \overline{\mathbb{R}}$ such that $F(D) = f(\mathrm{diag}(D)) = f(w)$, where diag is the adjoint of Diag
- $g := \nabla f(x)$, $G := \nabla F(D) = \mathrm{Diag}(\nabla f(x)) = \mathrm{Diag}(g)$
- $G := \mathrm{Diag}(g)$, $U_{\mathsf{p}} = G(G^\top G)^{-1/2} = (\mathrm{Diag}(g_i/|g_i|))_{1 \leqslant i \leqslant d} = \mathrm{Diag}(\mathrm{sgn}(g))$
- $\|\!|G|\!\|_{\mathrm{nuc}} = \sum_{i=1}^{d} |g_i| = \|g\|_1$
- Hence, PolarGrad on $D \Leftrightarrow$ unnormalized signSGD in its vector form:

$$D_{k+1} = D_k - \gamma_k \|g_k\|_1 \,\mathrm{Diag}(\mathrm{sgn}(g_k)) \quad \Leftrightarrow \quad x_{k+1} = x_k - \gamma_k \|g_k\|_1 \mathrm{sgn}(g_k)$$

Reduction of Matrices to Vectors or Scalars in PolarGrad and Muon

- In practice, Muon is only used for parameters of dimension $\geqslant 2$
- For vectors or scalars, Adam(W) is still used. Why?

Reduction of Matrices to Vectors or Scalars in PolarGrad and Muon

- In practice, Muon is only used for parameters of dimension $\geqslant 2$
- For vectors or scalars, Adam(W) is still used. Why?
- When X is a vector ($m = 1$ or $n = 1$ but not both), PolarGrad reduces to vanilla SGD whereas Muon without momentum reduces to ℓ_2-normalized SGD
- When X is a scalar ($m = n = 1$), PolarGrad again reduces to vanilla SGD whereas Muon without momentum reduces to signSGD
- Preconditioning is lost

Concluding Remarks

- A theoretically grounded explanation for the success of Muon through the lens of preconditioning
- A unifying preconditioning view of Muon and Adam in addition to the popular non-Euclidean steepest descent view: Hessian vs. gradient
- Introduced **PolarGrad**:
  - Benefit of the nuclear norm scaling term (null-gradient consistency)
  - Connection to polar decomposition
  - Choices of inexact polar oracles (NS vs. QDWH)
- Is matrix orthogonalization *optimal*?

Concluding Remarks

- A theoretically grounded explanation for the success of Muon through the lens of preconditioning
- A unifying preconditioning view of Muon and Adam in addition to the popular non-Euclidean steepest descent view: Hessian vs. gradient
- Introduced **PolarGrad**:
  - Benefit of the nuclear norm scaling term (null-gradient consistency)
  - Connection to polar decomposition
  - Choices of inexact polar oracles (NS vs. QDWH)
- Is matrix orthogonalization *optimal*?
- DNNs are trained in a BCD fashion (Zeng et al., 2019) with parallel updates:
  - Different optimizers for scalar, vector, matrix and tensor parameters
  - Different hyperparameters for matrix parameters of different sizes
  - Architecture-optimizer co-design?

Pre-prints

Thank you!

PolarGrad: *A Class of Matrix-Gradient Optimizers from a Unifying Preconditioning Perspective*
Tim Tsz-Kit Lau, Qi Long, and Weijie Su. arXiv:2505.21799

Follow-up Work by Prof. Weijie Su

Isotropic Curvature Model for Understanding Deep Learning Optimization: Is Gradient Orthogonalization Optimal?
Weijie Su. arXiv:2511.00674

References I

N. Amsel, D. Persson, C. Musco, and R. Gower. The Polar Express: Optimal matrix sign methods and their application to the Muon algorithm. *arXiv preprint arXiv:2505.16932*, 2025.

J. Bernstein and L. Newhouse. Modular duality in deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025.

J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2): 223–311, 2018.

D. Carlson, Y.-P. Hsieh, E. Collins, L. Carin, and V. Cevher. Stochastic spectral descent for discrete graphical models. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):296–311, 2016.

G. E. Dahl, F. Schneider, Z. Nado, N. Agarwal, C. S. Sastry, P. Hennig, S. Medapati, R. Eschenhagen, P. Kasimbeg, D. Suo, J. Bae, J. Gilmer, A. L. Peirson, B. Khan, R. Anil, M. Rabbat, S. Krishnan, D. Snider, E. Amid, K. Chen, C. J. Maddison, R. Vasudev, M. Badura, A. Garg, and P. Mattson. Benchmarking neural network training algorithms. *arXiv preprint arXiv:2306.07179*, 2023.

V. Gupta, T. Koren, and Y. Singer. Shampoo: Preconditioned stochastic tensor optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

K. Jordan, Y. Jin, V. Boza, Y. Jiacheng, F. Cecista, L. Newhouse, and J. Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL `https://kellerjordan.github.io/posts/muon/`.

References II

P. Kasimbeg, F. Schneider, R. Eschenhagen, J. Bae, C. S. Sastry, M. Saroufim, B. Feng, L. Wright, E. Z. Yang, Z. Nado, S. Medapati, P. Hennig, M. Rabbat, and G. E. Dahl. Accelerating neural network training: An analysis of the AlgoPerf competition. In *International Conference on Learning Representations (ICLR)*, 2025.

D. P. Kingma and J. L. Ba. Adam: a method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

J. Liu, J. Su, X. Yao, Z. Jiang, G. Lai, Y. Du, Y. Qin, W. Xu, E. Lu, J. Yan, Y. Chen, H. Zheng, Y. Liu, S. Liu, B. Yin, W. He, H. Zhu, Y. Wang, J. Wang, M. Dong, Z. Zhang, Y. Kang, H. Zhang, X. Xu, Y. Zhang, Y. Wu, X. Zhou, and Z. Yang. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Y. Nakatsukasa and R. W. Freund. Computing fundamental matrix decompositions accurately via the matrix sign function in two iterations: The power of Zolotarev's functions. *SIAM Review*, 58(3):461–493, 2016.

Y. Nakatsukasa and N. J. Higham. Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the SVD. *SIAM Journal on Scientific Computing*, 35(3):A1325–A1349, 2013.

Y. Nakatsukasa, Z. Bai, and F. Gygi. Optimizing Halley's iteration for computing the matrix polar decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31(5):2700–2720, 2010.

F. Orabona. Neural networks (maybe) evolved to make Adam the best optimizer, 2020. URL https://parameterfree.com/2020/12/06/neural-network-maybe-evolved-to-make-adam-the-best-optimizer/.

References III

H.-J. M. Shi, T.-H. Lee, S. Iwasaki, J. Gallego-Posada, Z. Li, K. Rangadurai, D. Mudigere, and M. Rabbat. A distributed data-parallel PyTorch implementation of the distributed Shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*, 2023.

N. Vyas, D. Morwani, R. Zhao, I. Shapira, D. Brandfonbrener, L. Janson, and S. Kakade. SOAP: Improving and stabilizing Shampoo using Adam. In *International Conference on Learning Representations (ICLR)*, 2025.

J. Zeng, T. T.-K. Lau, S. Lin, and Y. Yao. Global convergence of block coordinate descent in deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.