

LiMuon: Light and Fast Muon Optimizer for Large Models

Feihu Huang¹, Yuning Luo², Songcan Chen¹

1. Nanjing University of Aeronautics & Astronautics
2. National University of Singapore



Roadmap

- ❑ Background
- ❑ LiMuon Optimizer
- ❑ Convergence Properties
- ❑ Experimental Results
- ❑ Conclusions

Roadmap

- Background
- LiMuon Optimizer
- Convergence Properties
- Experimental Results
- Conclusions

Background

- ▶ Over the past decade, training large models has been dominated by the **vector-wise** optimization methods including SGD, momentum-based SGD, **Adam** and **AdamW** algorithms.



ChatGPT



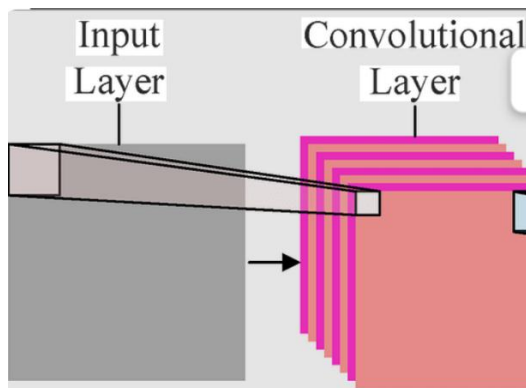
deepseek



Background

- These vector-wise algorithms **ignore the inherent matrix structure** of parameters in AI models such as the convolutional layers in **CNNs**, and the query, key, and value matrices in **transformers**, which results in suboptimal convergence efficiency.

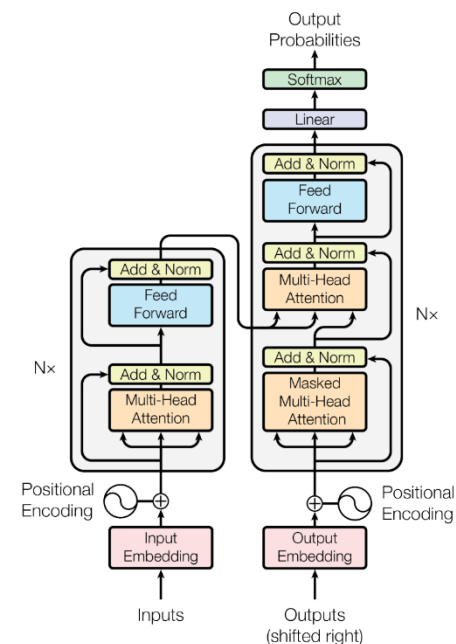
CNNs



$$Z^{(t)} = W^{(t)} * X^{(t-1)} + b^{(t)}$$

$$X^{(t)} = a(Z^{(t)})$$

Transformers



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Background

- More recently, a useful matrix-wise optimizer(i.e., [Muon](#)) has been proposed to train the large matrix-structured models, which shows markedly faster convergence than the vector-wise algorithms.

$$\min_{W \in \mathbb{R}^{m \times n}} F(W) = \mathbb{E}_{\xi \sim \mathcal{D}}[f(W; \xi)]$$

Algorithm 1 Muon Optimizer [Jordan et al.](#), [2024](#)

```
1: Input:  $\eta > 0$  and  $\mu > 0$ ;  
2: Initialize:  $W_1 \in \mathbb{R}^{m \times n}$ ;  
3: for  $t = 1, 2, \dots, T$  do  
4:   Draw a sample  $\xi_t \sim \mathcal{D}$ ;  
5:    $G_t = \nabla f(W_t; \xi_t)$ ;  
6:    $M_t = \mu M_{t-1} + G_t$ ;  
7:    $O_t = \text{Newton-Schulz}(M_t)$ ;  
8:    $W_{t+1} = W_t - \eta O_t$ ;  
9: end for  
10: Output:  $W_T$ .
```

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

Background

- For example, the [Muon](#) optimizer has been used to train some typical large models including [DeepSeek V4](#), [Kimi K2](#) and [YOLO26](#).



Background

- ▶ However, the Muon optimizer still requires **high sample complexity** and **high memory cost** in training large models.



more Compute/GPU
more Electricity



more Storage



Expensive!



Roadmap

- Background
- LiMuon Optimizer
- Convergence Properties
- Experimental Results
- Conclusions

LiMuon Optimizer

In our paper, we propose a light and fast Muon (called as **LiMuon**) optimizer, which **simultaneously** has a **lower memory cost** and **lower sample complexity** than the Muon and its variants.

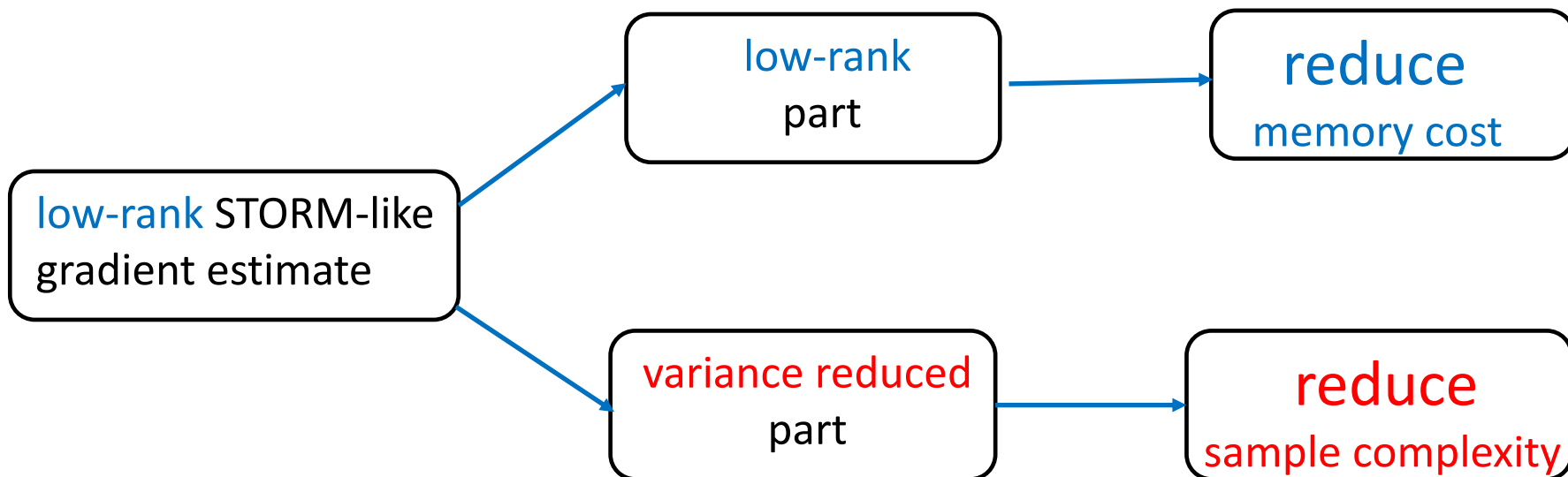
$$\min_{W \in \mathbb{R}^{m \times n}} F(W) = \mathbb{E}_{\xi \sim \mathcal{D}}[f(W; \xi)]$$

LiMuon Optimizer

Here, we first introduce our **low-rank** STORM-like gradient estimate:

$$M_{t+1} = \nabla f(W_{t+1}; \xi_{t+1}) + (1 - \beta_{t+1})(\hat{M}_t - \nabla f(W_t; \xi_{t+1}))$$

where $\hat{M}_t = \hat{U}_t \hat{S}_t \hat{V}_t^\top$ is a low-rank approximated estimate of momentum M_t .



LiMuon Optimizer

Algorithm 1 LiMuon Optimizer

- 1: **Input:** $\eta_t > 0$, $\beta_t \in [0, 1)$, $0 < \hat{r} < r = \min(m, n)$ and $s \geq 2$;
 - 2: **Initialize:** $W_0 \in \mathbb{R}^{m \times n}$, $\xi_0 \sim \mathcal{D}$ and $M_0 = \nabla f(W_0; \xi_0)$;
 - 3: **for** $t = 0, 1, \dots, T - 1$ **do**
 - 4: $(U_t, \Sigma_t, V_t) = \text{SVD}(M_t)$;
 - 5: $W_{t+1} = W_t - \eta_t U_t V_t^\top$;
 - 6: Draw a sample $\xi_{t+1} \sim \mathcal{D}$;
 - 7: $M_{t+1} = \nabla f(W_{t+1}; \xi_{t+1}) + (1 - \beta_{t+1})(M_t - \nabla f(W_t; \xi_{t+1}))$ (**Option #1**);
 - 8: $(\hat{U}_t, \hat{S}_t, \hat{V}_t) = \text{RSVD}(M_t, \hat{r}, s)$ (from Algorithm 2), and $\hat{M}_t = \hat{U}_t \hat{S}_t \hat{V}_t^\top$;
 - 9: $M_{t+1} = \nabla f(W_{t+1}; \xi_{t+1}) + (1 - \beta_{t+1})(\hat{M}_t - \nabla f(W_t; \xi_{t+1}))$ (**Option #2**).
 - 10: **end for**
 - 11: **Output:** W_T .
-

Randomized SVD

Algorithm 2 RSVD($A, \hat{r}, s$)

- 1: **Input:** Matrix $A \in \mathbb{R}^{m \times n}$, target rank $\hat{r} > 0$, oversampling parameter $s > 0$;
 - 2: **Output:** U, Σ, V ;
 - 3: $l = \hat{r} + s$, and generate a random Gaussian matrix $\Omega \in \mathbb{R}^{n \times l}$;
 - 4: $Y = A\Omega \in \mathbb{R}^{m \times l}$, and compute the QR decomposition $Y = QR$;
 - 5: $B = Q^\top A \in \mathbb{R}^{l \times n}$, and compute SVD of the small matrix: $(\tilde{U}, \Sigma, V) = \text{SVD}(B)$, and $U = Q\tilde{U}$.
-

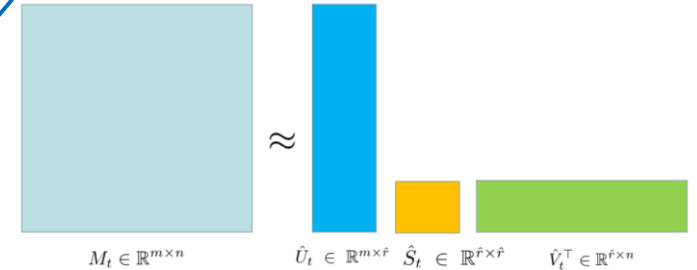


Figure 1. Low Rank Compression of Momentum M_t .

LiMuon Optimizer

Algorithm 3 LiMuon Optimizer with Newton-Schulz

- 1: **Input:** $\eta_t > 0$, $\beta_t \in [0, 1)$, $0 < \hat{r} < r = \min(m, n)$, $s \geq 2$ and $q \geq 1$;
 - 2: **Initialize:** $W_0 \in \mathbb{R}^{m \times n}$, $\xi_0 \sim \mathcal{D}$ and $M_0 = \nabla f(W_0; \xi_0)$;
 - 3: **for** $t = 0, 1, \dots, T - 1$ **do**
 - 4: $O_t = \text{Newton-Schulz}(M_t, q)$ (from Algorithm 4);
 - 5: $W_{t+1} = W_t - \eta_t O_t$;
 - 6: Draw a sample $\xi_{t+1} \sim \mathcal{D}$;
 - 7: $M_{t+1} = \nabla f(W_{t+1}; \xi_{t+1}) + (1 - \beta_{t+1})(M_t - \nabla f(W_t; \xi_{t+1}))$ (**Option #1**);
 - 8: $(\hat{U}_t, \hat{S}_t, \hat{V}_t) = \text{RSVD}(M_t, \hat{r}, s)$ (from Algorithm 2), and $\hat{M}_t = \hat{U}_t \hat{S}_t \hat{V}_t^T$;
 - 9: $M_{t+1} = \nabla f(W_{t+1}; \xi_{t+1}) + (1 - \beta_{t+1})(\hat{M}_t - \nabla f(W_t; \xi_{t+1}))$ (**Option #2**).
 - 10: **end for**
 - 11: **Output:** W_T .
-

Algorithm 4 Newton-Schulz(M, q)

- 1: **Input:** Matrix $M \in \mathbb{R}^{m \times n}$, step size $q \geq 1$, and polynomial $p_\kappa(\cdot)$, where κ is degree;
 - 2: **Initialize:** $X_0 = M/\alpha$ with $\alpha = \max(1, \|M\|_F)$;
 - 3: **for** $j = 1, 2, \dots, q$ **do**
 - 4: $X_j = p_\kappa(X_{j-1} X_{j-1}^\top) X_{j-1}$;
 - 5: **end for**
 - 6: $O = X_q$;
 - 7: **Output:** O .
-

We use a small and fixed number of **Newton-Schulz steps** to approximate the orthogonalization process **instead of computing exact SVD**.

Roadmap

- Background
- LiMuon Optimizer
- Convergence Properties
- Experimental Results
- Conclusions

Convergence Properties

In our paper, we studied the convergence properties of our LiMuon optimizer with **exact SVD** and **Newton-Schulz steps**, **respectively**, under the **generalized smoothness condition**.

Table 1. Comparison of the Muon optimizer and its variants for finding an ϵ -stationary point ($\mathbb{E}\|\nabla F(W)\|_F \leq \epsilon$) of nonconvex stochastic optimization problem (1). Here $\hat{r} \ll \min(m, n)$ and $\chi_q = \frac{1}{1-\varepsilon_q} > 1$, where $\varepsilon_q \in (0, 1)$ is a polar approximation error.

Algorithm	Reference	SFO Complexity	State Memory	Generalized Smoothness	Newton-Schulz
Muon	(Shen et al., 2025)	$O(\epsilon^{-4})$	mn		
SCG	(Pethick et al., 2025a)	$O(\epsilon^{-4})$	mn		
Gluon	(Riabinin et al., 2025)	$O(\epsilon^{-4})$	mn	✓	
GGNC	(Pethick et al., 2025b)	$O(\epsilon^{-4})$	mn	✓	
SUMO	(Refael et al., 2025)	$O(\epsilon^{-4})$	$(m+n)\hat{r}$		
Muon ⁺⁺	(Sfyraki & Wang, 2025)	$O(\epsilon^{-3})$	mn		
LiMuon	Ours	$O(\epsilon^{-3})$	$(m+n)\hat{r}$	✓	
Muon	(Kim & Oh, 2026)	$O(\chi_q^4 \epsilon^{-4})$	mn		✓
LiMuon	Ours	$O(\chi_q^3 \epsilon^{-3})$	$(m+n)\hat{r}$	✓	✓

Convergence Properties

Assumption 4.4 (Frobenius-Norm Generalized Smoothness). Functions $F(W)$ and $f(W; \xi)$ for all $\xi \sim \mathcal{D}$ are (L_0, L_1) -Frobenius norm smooth, if for any $W, W' \in \mathbb{R}^{m \times n}$, we have

$$\begin{aligned} & \|\nabla F(W) - \nabla F(W')\|_F^2 \\ & \leq (L_0^2 + L_1^2 \|\nabla F(W)\|_F^2) \|W - W'\|_F^2, \\ & \mathbb{E} \|\nabla f(W; \xi) - \nabla f(W'; \xi)\|_F^2 \\ & \leq (L_0^2 + L_1^2 (\mathbb{E} \|\nabla F(W)\|_F^2)) \|W - W'\|_F^2. \end{aligned}$$

Theorem 4.13. Under the Assumptions 4.4, 4.5, 4.6, the sequence $\{W_t\}_{t=0}^T$ be generated from Algorithm 1 with **Option #2**. Let $\beta_t = \beta \in (0, 1)$ and $\eta_t = \eta \leq \min\left(\frac{1}{4L_1r}, \frac{\beta}{12L_1r(1-\beta)}\right)$ for all $t \geq 0$, and $s \geq 2$, $\hat{r} \geq 2$, $s + \hat{r} \leq r$, and $\gamma \leq \frac{\beta}{3\sqrt{r}(1-\beta)}$, we have

$$\begin{aligned} \frac{1}{T+1} \sum_{t=0}^T \mathbb{E} \|\nabla F(W_t)\|_* & \leq \frac{4(F(W_0) - F^*)}{T\eta} + 2rL_0\eta \\ & + \frac{6\sqrt{r}\sigma}{T\beta} + 6\sqrt{r} \sqrt{\frac{\beta}{2-\beta}} \sigma + 6L_0\eta r \sqrt{\frac{(1-\beta)^2}{(2-\beta)\beta}}, \end{aligned}$$

where $\gamma = \tau \left(1 + \frac{\hat{r}}{s-1}\right)^{\frac{1}{2}}$ and $\tau \in (0, 1)$.

Remark 4.14. From the above Theorem 4.13, let $\eta = O(\frac{1}{T^{2/3}})$ and $\beta = O(\frac{1}{T^{2/3}})$, we have

$$\frac{1}{T+1} \sum_{t=0}^T \mathbb{E} \|\nabla F(W_t)\|_* \leq O\left(\frac{1}{T^{1/3}}\right) \leq \epsilon.$$

Our LiMuon algorithm with lower memory cost still obtains a lower sample complexity of $O(\epsilon^{-3})$ under generalized smooth condition.

Convergence Properties

Assumption 4.16 (Nuclear-Norm Generalized Smoothness). Functions $F(W)$ and $f(W; \xi)$ for all $\xi \sim \mathcal{D}$ are (L_0, L_1) -nuclear norm smooth, if for any $W, W' \in \mathbb{R}^{m \times n}$, we have

$$\begin{aligned} & \|\nabla F(W) - \nabla F(W')\|_*^2 \\ & \leq (L_0^2 + L_1^2 \|\nabla F(W)\|_*^2) \|W - W'\|_{op}^2, \\ & \mathbb{E} \|\nabla f(W; \xi) - \nabla f(W'; \xi)\|_*^2 \\ & \leq (L_0^2 + L_1^2 (\mathbb{E} \|\nabla F(W)\|_*)^2) \|W - W'\|_{op}^2. \end{aligned}$$

Theorem 4.23. Under the Assumptions 4.16, 4.5, 4.6, the sequence $\{W_t\}_{t=0}^T$ be generated from Algorithm 3 with **Option #2**. Let $\beta_t = \beta \in (0, 1)$, and $\eta_t = \eta \leq \min(\frac{1-\varepsilon_q}{4L_1(1+\varepsilon_q)^2}, \frac{(1-\varepsilon_q)\beta}{4L_1\sqrt{r}(1-\beta)(1+\varepsilon_q)(3+\varepsilon_q)})$ for all $t \geq 0$, $s \geq 2$, $\hat{r} \geq 2$, $\hat{r} + s \leq r$ and $\gamma \leq \frac{(1-\varepsilon_q)\beta}{(3+\varepsilon_q)\sqrt{r}(1-\beta)}$, we have

$$\begin{aligned} & \frac{1}{T+1} \sum_{t=0}^T \mathbb{E} \|\nabla F(W_t)\|_* \leq \frac{4(F(W_0) - F^*)}{(T+1)\eta\chi_q^{-1}} \\ & + \frac{2L_0\eta(1+\varepsilon_q)^2}{\chi_q^{-1}} + \frac{2(3+\varepsilon_q)}{\chi_q^{-1}} \left(\frac{\sqrt{r}\sigma}{(T+1)\beta} + \sqrt{r} \sqrt{\frac{\beta}{2-\beta}} \sigma \right. \\ & \left. + \sqrt{r} \sqrt{\frac{(1-\beta)^2}{(2-\beta)\beta}} L_0\eta(1+\varepsilon_q) \right), \end{aligned}$$

where $\gamma = \tau \left(1 + \frac{\hat{r}}{s-1}\right)^{\frac{1}{2}}$ with $\tau \in (0, 1)$, and $\chi_q = \frac{1}{1-\varepsilon_q} > 1$.

Remark 4.24. From the above Theorem 4.23, let $\eta = O(\frac{1}{T^{2/3}})$ and $\beta = O(\frac{1}{T^{2/3}})$, we have

$$\frac{1}{T+1} \sum_{t=0}^T \mathbb{E} \|\nabla F(W_t)\|_* \leq O\left(\frac{\chi_q}{T^{1/3}}\right) \leq \epsilon.$$

Our LiMuon algorithm with lower memory cost still obtains a lower sample complexity of $O(\chi_q^3 \epsilon^{-3})$ under generalized smooth condition than $O(\chi_q^4 \epsilon^{-4})$ of the Muon optimizer with Newton-Schulz (Kim & Oh, 2026).

Roadmap

- Background
- LiMuon Optimizer
- Convergence Properties
- **Experimental Results**
- Conclusions

Experimental Results

1) Training Mamba-130M Model

Table 2. Performance comparison at Mamba-130M

Method	Memory (GB) ↓	Training PPL ↓	Validation PPL ↓
Adam	22.92	480.33	526.19
AdamW	22.92	253.75	266.43
Lion	22.56	75.25	83.82
SUMO	20.10	111.56	118.52
Muon	22.20	62.49	71.27
Muon++	22.35	51.97	56.79
LiMuon(rank=4)	18.86	71.46	79.12
LiMuon(rank=8)	20.25	59.07	62.23
LiMuon(rank=16)	22.31	50.26	53.84
LiMuon(full rank)	22.80	47.08	47.78

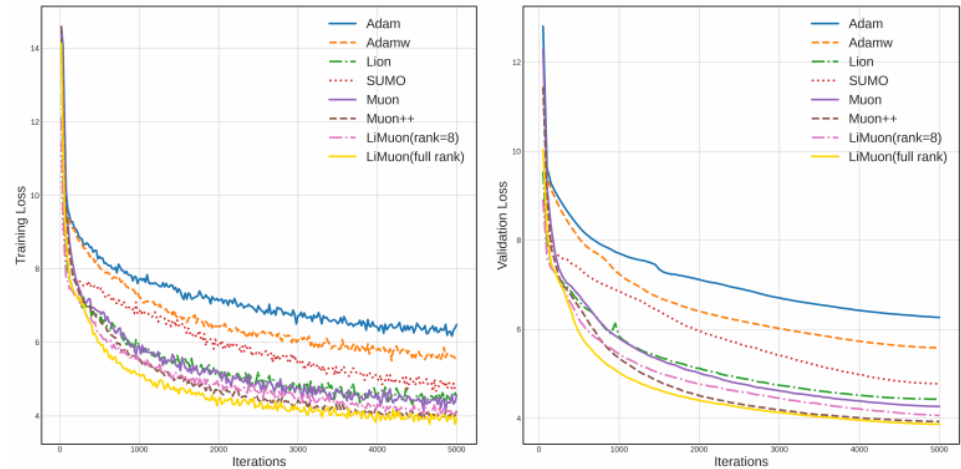


Figure 2. Performance comparison at Mamba-130M

Experimental Results

2) Training Qwen2.5-0.5B Model

From these results, the full-rank LiMuon achieves the lowest perplexity (PPL), while providing more stable training dynamics. Notably, our low-rank LiMuon variant (rank= 8) achieve competitive performance with reduced memory cost, making our LiMuon suitable for training larger models under memory cost constraints.

Table 3. Performance comparison at Qwen-2.5-0.5B

Method	Memory (GB) ↓	Training PPL ↓	Validation PPL ↓
Adam	55.24	91.84	181.27
AdamW	55.26	57.97	113.25
Lion	54.08	48.42	113.92
SUMO	52.73	41.26	88.73
Muon	54.14	35.52	67.60
Muon++	54.30	35.16	82.26
LiMuon(rank=4)	50.86	41.46	92.06
LiMuon(rank=8)	52.55	38.86	54.87
LiMuon(rank=16)	54.21	33.43	46.77
LiMuon(full rank)	55.15	30.87	40.83

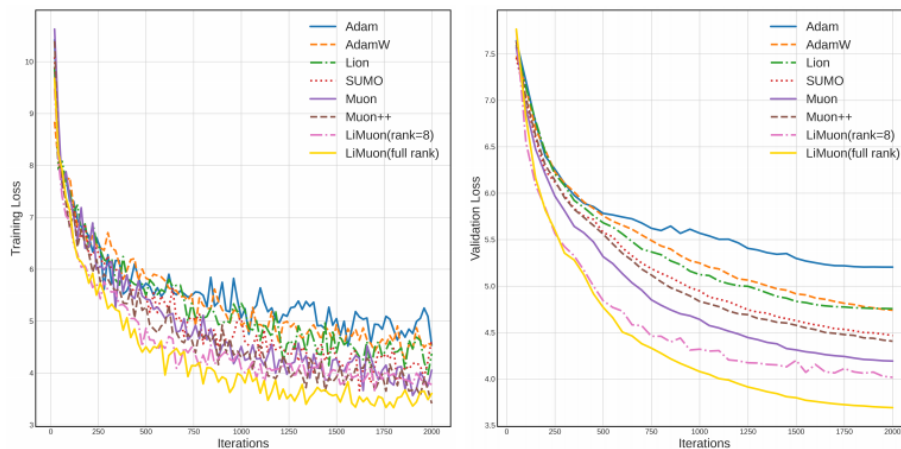


Figure 3. Performance comparison at Qwen-2.5-0.5B.

Experimental Results

3) Training ViT Model

Table 4. Performance comparison at ViT

Method	Memory (GB) ↓	Training Top-1 Accuracy(%) ↑	Training Top-5 Accuracy(%) ↑	Validation Top-1 Accuracy(%) ↑	Validation Top-5 Accuracy(%) ↑
Adam	5.57	37.47	65.45	32.91	59.96
AdamW	5.58	38.41	65.56	35.85	63.96
Lion	5.44	40.97	68.55	40.21	68.02
SUMO	5.31	51.23	78.08	44.23	70.19
Muon	5.50	78.96	93.64	47.87	73.17
Muon++	5.52	60.44	83.52	43.62	69.94
LiMuon (rank=4)	5.10	64.73	85.19	44.35	70.86
LiMuon (rank=8)	5.28	71.94	90.06	46.75	72.17
LiMuon (rank=16)	5.44	80.58	93.91	47.98	73.22
LiMuon (full rank)	5.53	81.99	94.64	48.04	73.58

Experimental Results

3) Training ViT Model

In Table 5, Time. Step is the wall-clock time of updating variable, which does not include time of computing gradients via backpropagation. Time to Target is the wall-clock time to first reach a target loss, which include time of computing gradients via backpropagation. Here we give a target loss $L^* = 3.70$ (an example, the other target losses have similar conclusions). From Table 5, our full rank LiMuon uses the shortest amount of time to reach the target loss. Meanwhile, our low-rank LiMuon (rank=8) uses a comparable time to reach the target loss under less memory usage.

Table 5. Runtime analysis on ViT.

Optimizer	Time. Step ↓	Time to Target ↓
<i>Baseline optimizers</i>		
Adam	5.58ms	15min15s
AdamW	5.66ms	13min11s
Lion	4.81ms	12min37s
Muon	23.43ms	7min6s
Muon++	24.50ms	9min54s
SUMO	22.08ms	11min20s
<i>LiMuon (ours)</i>		
LiMuon ($\hat{r}=4$)	24.75ms	12min4s
LiMuon ($\hat{r}=8$)	25.49ms	10min29s
LiMuon ($\hat{r}=16$)	26.87ms	10min51s
LiMuon (full rank)	23.37ms	7min 1s

Roadmap

- Background
- LiMuon Optimizer
- Convergence Properties
- Experimental Results
- Conclusions

Conclusions

- ▶ 1) We proposed a LiMuon optimizer, which **simultaneously** has a **lower memory cost** and **lower sample complexity** than the Muon and its variants.
- ▶ 2) We studied the convergence properties of our LiMuon optimizer with **exact SVD** and **Newton-Schulz steps**, **respectively**, under the generalized smoothness condition.
- ▶ 3) In particular, we proved that our LiMuon with Newton-Schulz steps obtains a lower sample complexity under a lower memory cost than the Muon with Newton-Schulz steps.

Thanks!

Q&A