Preconditioning and Normalization in Optimizing Deep Neural Networks

Qinzi Zhang Ashok Cutkosky

February 11, 2025

1 Overview

We introduce a new algorithm for optimizing neural networks. Our method combines ideas from several recent (and less recent) optimization algorithms to produce a single optimizer that appears to outperform the current state-of-the-art. For those familiar with the current trends in the literature, the scheme is very simple. For each tensor W in the weights of a neural network, we do the following:

1. Decide upon a norm $\|\cdot\|$ appropriate for that tensor (more on how we do this latter), so that we build a global norm formed by the max of the individual tensor norms, as described by [1, 2].
2. For each tensor W, compute the following "LaProp" update [3], a variant of the standard Adam [4] update:
$$v_t = v_{t-1}\beta_2 + (1-\beta_2)g_t^2$$
$$m_t = m_{t-1}\beta_1 + (1-\beta_1)g_t/\sqrt{v_t}$$
where g_t is the gradient with respect to W.
3. Identify the vector δ_t that satisfies $\|\delta_t\| = 1$ and $\langle \delta_t, m\rangle = \|m\|_\star$, where $\|\cdot\|_\star$ is the dual norm to $\|\cdot\|$.
4. Update the tensor W via $W \leftarrow W - \eta_t\delta$ where $\eta_t \in \mathbb{R}$ is a learning rate.

To see where this procedure comes from, let us consider how it connects to three prior algorithms.

- First, it is clear that if we skip step (3) and simply set $\delta_W = m_W$, then we recover the LaProp variant of the standard Adam update.
- Next, suppose we replace step (2) with the standard SGDM update: $m_t = m_{t-1}\beta_1 + (1-\beta_1)g_t$. In this case, the final update δ depends on the norm $\|\cdot\|$. If $\|\cdot\|$ is the ordinary Euclidean 2-norm (also called the Frobenius norm for matrices), we recover normalized SGD. On the other hand, if $\|\cdot\|$ is instead the spectral or operator norm of a matrix $\|W\| = \sup_{\|v\|_2=1}\|Wv\|_2$, then the update becomes exactly the Muon optimizer that has recently demonstrated the fastest performance on the "gpt speed running" task [5].
- Finally, if we go back to setting m_t via the Adam update, and set $\|\cdot\|$ to be the 2-norm then we recover the LAMB algorithm [6], although without scaling the learning rate for each layer by the norm of the layer.

So, our hypothesis is that the choice precondition-followed-by-normalization scheme employed by LAMB, and the normalize-by-spectral-norm scheme without preconditioning employed by Muon are orthogonally advantageous, and that combining the two can yield improvements. To preview our current empirical results, we find that performing the preconditioned and normalized update using the spectral norm on all the internal weights of the network improves the performance over our Muon baseline. Moreover, by employing 2-norm normalization on bias updates as well as spectral normalization on the head layer, we see further improvements.

We also employ an additional "offset momentum" in our results that results in minor improvements, which we will describe later in this report.

2 Background

We employ the standard iterative stochastic optimization setup: at each of T time steps, our algorithm produces a set of weights $W_t = (W_t^1, \ldots, W_t^N)$ where N is the number of tensors in our model. Then, we receive a gradient $g_t = (g_t^1, \ldots, g_t^N)$ where $g_t = \nabla \ell(W_t, x_t)$ where ℓ is a loss function and x_t is the t-th mini-batch of examples. The algorithm then uses g_t to update some internal optimizer state and produce W_{t+1}. We use $\|\cdot\|_2$ to indicate the 2-norm (Frobenious norm for matrices), and $\|\cdot\|_{\text{op}}$ to indicate the operator norm: for any matrix M: $\|M\|_{\text{op}} = \sup_{\|v\|_2=1} \|Mv\|_2$. We did not consider operator norms where the v and output Mv are measured in different norms.

2.1 Adam

The standard algorithm for optimizing deep neural networks in this paradigm is Adam [4]. Adam differs from SGD in two main ways: incorporation of momentum [7], as well as diagonal preconditioning inspired by similar techniques used in online convex optimization (e.g. the AdaGrad algorithm) [8, 9]. Roughly speaking, the update for each W^i is as follows (dropping the i superscript to avoid clutter):

$$\begin{aligned} m_t &= m_{t-1}\beta_1 + (1-\beta_1)g_t \\ v_t &= v_{t-1}\beta_2 + (1-\beta_2)g_t^2 \\ W_{t+1} &= W_t - \eta_t \frac{m_t}{\sqrt{v_t}} \end{aligned}$$

where all multiplication and division of tensors above are computed elementwise, and we have removed the bias-correction terms for simplicity of presentation.

Our update will instead use a variant of the Adam update more reminiscent of LaProp [3]. This made some aspects of the implementation cleaner; it is worthwhile to test the Adam update exactly in our framework. The LaProp update is the following:

$$\begin{aligned} v_t &= v_{t-1}\beta_2 + (1-\beta_2)g_t^2 \\ m_t &= m_{t-1}\beta_1 + (1-\beta_1)g_t/\sqrt{v_t} \\ W_{t+1} &= W_t - \eta_t m_t \end{aligned}$$

2.2 LAMB

The LAMB algorithm [6] proposed to normalize the Adam update for certain internal tensors (i.e. usually only for internal weight *matrices* rather than bias or scale vectors). The algorithm proposed two main changes: first, compute the Adam update $\frac{m_t}{\sqrt{v_t}}$, but then normalize it to obtain $\delta_t = \frac{m_t/\sqrt{v_t}}{\|m_t/\sqrt{v_t}\|_2}$. Notice that we can equivalently write this operation as $\delta_t = \text{argmax}_{\|\delta\|_2=1}\langle \delta, m_t/\sqrt{v_t}\rangle$. Next, they propose to scale the learning rate η_t by $\|W_t\|$. The empirical advantages of LAMB were later challenged by [10], who suggest that much of the improvement attributed to LAMB can be recovered by appropriately tuning Adam. Nevertheless, the approach of normalizing the Adam update forms a core of our technique.

2.3 Shampoo

Recent advances to improve Adam have often focused on improving the preconditioning method by using more advanced non-diagonal preconditioners. One successful such approach is the Shampoo algorithm [11], which utilizes the same online optimization framework that drives AdaGrad to derive the optimal preconditioner for subject to the condition that the preconditioned update for a gradient of shape $g_t^i \in \mathbb{R}^{m\times n}$ has the form $g_t^i \mapsto L g_t^i R$ for some matrices $L \in \mathbb{R}^{m\times m}$ and $R \in \mathbb{R}^{n\times n}$. Formally, one should set $L \propto \left(\sum_{t=1}^T g_t^i (g_t^i)^\top / T\right)^{-1/4} \approx \mathbb{E}[g_t^i (g_t^i)^\top]^{-1/4}$ while $R \propto \left(\sum_{t=1}^T (g_t^i)^\top g_t^i\right)^{-1/4} \approx \left(\mathbb{E}[(g_t^i)^\top g_t^i]\right)^{-1/4}$.

Since its initial description, the Shampoo algorithm has been refined in a number of ways, including introducing better ways to compute the relevant L and R matrices, or modifications to the formulas used to produce L and R such as replacing the sums with exponential moving averages, or modifying the exponent

from $-1/4$ to $-1/2$ [12]. The resulting optimizer is very effective; the DistributedShampoo implemention recently won the MLCommons AlgoPerf optimizer competition [13].

2.4 SOAP

More recently, the SOAP algorithm [14] as an effort to combat some of the numerical issues that complicate Shampoo implementions. Computing the L and R matrices in Shampoo is notoriously complicated and expensive: they are usually only updated every few iterations. The SOAP optimizer computes the eigenvectors of the L and R matrices produced by Shampoo, performs a change-of-basis to these coordinates, computes an Adam update in this new basis, and then transforms back. This algorithm demonstrated improvement over Shampoo because it allows for the Adam update to be computed every iteration even if the L and R matrices are only updated occasionally.

2.5 Muon

Even more recently, the Muon optimizer [5] provided further improvements via a somewhat orthogonal technique: run a standard Adam update on all layers of a network *except* the internal weight matrices. For an internal weight matrix W^i, first compute the momentum buffer produced by Adam, m_t^i, and then compute $\delta_t = \text{argmax}_{\|\delta\|_{\text{op}}=1}\langle \delta_t, m_t\rangle$[1] The final update is then $W_{t+1}^i \leftarrow W_t^i - \eta_t \delta_t^i$.

2.6 Normalized SGD with arbitrary norm

Notice that, when using the 2-norm, solving the equation:

$$\delta = \text{argmin}\langle \delta, g\rangle + \frac{\lambda}{2}\|\delta\|^2$$

results in $\delta = -g/\lambda$, so that we recover gradient-descent-like algorithms.

If we want a normalized gradient descent, it is more natural to instead solve the equation:

$$\delta = \underset{\|\delta\|\leq\eta}{\text{argmin}}\langle \delta, g\rangle$$

which is equivalent to solving $\text{argmin}_{\|\delta\|\leq 1}\langle \delta, g\rangle$ and then multiplying by η. At a high level, this is essentially saying the update should attempt to decrease the loss as much as possible given the current gradient information, but avoid moving too much. This can be interpreted from a "trust-region" kind of viewpoint. We adopt this approach to deriving our normalized updates.

This viewpoint also has the advantage that we can naturally encompass approximate solutions to the above optimization: so long as $\langle \delta, g\rangle$ is within a constant factor of the optimal value and $\|\delta\|$ is within a constant factor of 1, previous normalized SGD analysis (e.g. [15]) will still apply.

3 Algorithm Intuition

To summarize the background, let us consider a particular tensor W corresponding internal weight matrix of the model (i.e. not an embedding matrix, or a normalization weight or a bias). The LAMB update for this tensor has the form (ignoring scaling by the weight matrix norm):

$$\begin{aligned} m_t^i &= m_{t-1}^i\beta_1 + (1-\beta_1)g_t^i/\sqrt{v_t^i} \\ \delta_t^i &= \underset{\|\delta\|_2=1}{\text{argmax}}\langle \delta, m_t^i\rangle \\ W_{t+1}^i &= W_t^i - \eta_t\delta_t^i \end{aligned}$$

[1] The original Muon description describes δ_t as the solution to a different optimization problem. Our formulation is equivalent, but makes it easier for us to connect the update to prior literature.

where here we have included the superscript i to specify which tensor is being considered. The Muon update has the form:

$$\begin{aligned} m_t^i &= m_{t-1}^i \beta_1 + (1-\beta_1) g_t^i \\ \delta_t^i &= \operatorname*{argmax}_{\|\delta\|_{\text{op}}=1} \langle \delta, m_t^i \rangle \\ W_{t+1}^i &= W_t^i - \eta_t \delta_t^i \end{aligned}$$

Our first suggestion is simply to combine these:

$$\begin{aligned} m_t^i &= m_{t-1}^i \beta_1 + (1-\beta_1) g_t^i / \sqrt{v_t^i} \\ \delta_t^i &= \operatorname*{argmax}_{\|\delta\|_{\text{op}}=1} \langle \delta, m_t^i \rangle \\ W_{t+1}^i &= W_t^i - \eta_t \delta_t^i \end{aligned}$$

We find that this already improves the performance over Muon (see Section 5).

Next, following the intuition of [1], we consider the possibility that we should use different norms for different tensors. Thus, if we use e.g. 2-norm for a particular tensor, we would recover the LAMB-style update, while the operator norm provides the "preconditioned-muon" update suggested above, and using the ∞-norm would suggest a kind of preconditioned "sign-SGD":

$$\begin{aligned} m_t^i &= m_{t-1}^i \beta_1 + (1-\beta_1) g_t^i / \sqrt{v_t^i} \\ \delta_t^i &= \operatorname*{argmax}_{\|\delta\|_{\text{inf}}=1} \langle \delta, m_t^i / \sqrt{v_t^i} \rangle \\ &= \operatorname{sign}(m_t^i / \sqrt{v_t^i}) \\ W_{t+1}^i &= W_t^i - \eta_t \delta_t^i \end{aligned}$$

So, how should we select these norms for different tensors? We can of course simply try different possible values, but there is some guiding intuition. We let $\|\cdot\|^i$ indicate the norm for a tensor W^i, and we propose the following:

if $x \in \mathbb{R}^d$ is the value of an embedding in a particular layer of the network and $f(x, W^i)$ the value after applying the transformation specified by a tensor W^i (e.g. matrix multiply, or bias addition, or per-coordinate scaling), then choose the "operator norm": $\|\delta\|^i = \sup_{\|x\|_2=1} \|f(x, W^i + \delta) - f(x, W^i)\|_2$ (recall that $\|\cdot\|_2$ is the 2-norm/Frobenius norm) For example:

- If W is a weight matrix so that $f(x, W) = Wx$, then $\|\delta\|^i = \|\delta\|_{\text{op}}$,
- If $f(x, W) = x + W$ (i.e. a bias), then $\|\delta\|^i = \|\delta\|_2$.
- If W is a scaling vector so that $f(x, W) = x \cdot W$ where $\cdot$ indicates coordinate-wise multiplication, then $\|\delta\|^i = \|\delta\|_\infty$

The overall norm for the full weight $W = (W^1, \ldots, W^N)$ is then $\sum_{i=1}^N \|W^i\|^i$. The final update can be described by setting:

$$\begin{aligned} m_t^i &= m_{t-1}^i \beta_1 + (1-\beta_1) g_t / \sqrt{v_t} \\ \delta_t &= \operatorname*{argmax}_{\|\delta\|=1} \langle \delta, m_t \rangle \\ W_{t+1} &= W_t - \eta_t \delta_t \end{aligned}$$

3.1 Simpler LR tuning

We found that by selecting the norm for each layer properly, we were able to use a single global learning rate rather than having a separate learning rate for each tensor.

3.2 adding Nesterov Momentum

We incorporate a modification inspired by Nesterov's momentum [16] as follows.

$$
\begin{aligned}
v_t &= v_{t-1} \cdot \beta_2 + g_t^2(1-\beta_2) \\
m_t &= m_{t-1} \cdot \beta_1 + g_t/\sqrt{v_t} \\
\Delta_t &= \operatorname*{argmax}_{\|\Delta\|=1} \langle \Delta, m_t\beta_1 + g_t/\sqrt{v_t} \rangle \\
W_{t+1} &= W_t - \eta_t \Delta_t
\end{aligned}
$$

3.3 Omitting Normalization

On tensors for which using the ∞-norm was the best-performing norm (which creates a sign-SGD-like update), we found that ommiting the normalization and simply use the Adam/LaProp update resulted in a slight performance improvement. It is perhaps natural to view diagonal preconditioning as measuring performance in the ∞-norm, as this is setting in which diagonal preconditioning strictly outperforms simple scalar learning rates in the online optimization setting. We conjecture from this that there is some better non-normalized update corresponding to the other norms as well.

4 Offset Momentum

In addition to the normalized update we describe above, we also employ a final modification to the update that has a momentum-like flavor. This change results in a relatively small improvement to the update.

Roughly speaking, [17] shows that for an optimizer that outputs updates $W_{t+1} = W_t + \Delta_t$ in response to gradients g_t, we can often improve the performance of the optimizer on strongly-convex losses by considering the modified optimizer:

$$
\begin{aligned}
R_t &= \sum_{i=1}^{t} \eta_t \Delta_t \\
O_t &= \frac{1}{t} \sum_{i=1}^{t} W_i \\
W_{t+1} &= R_t + O_t
\end{aligned}
$$

We make an exponentially weighted average version of this update as follows:

$$
\begin{aligned}
R_t &= \sum_{i=1}^{t} \eta_t \Delta_t \\
O_t &= \beta O_{t-1} + (1-\beta) W_{t-1} \\
W_{t+1} &= R_t + O_t
\end{aligned}
$$

After some rearrangement, this can be re-expressed in the form:

$$
\begin{aligned}
O_t &= \beta_{\text{offset}} O_{t-1} + (1-\beta_{\text{offset}}) \eta_t \Delta_t \\
W_{t+1} &= W_{t-1} + (O_t + \Delta_t)
\end{aligned}
$$

So that the final update looks nearly identical to the Nesterov momentum update. We apply this on top of our final update (after applying learning rate schedule).

So, overall our update has the form:

$$
\begin{aligned}
v_t &= v_{t-1} \cdot \beta_2 + g_t^2(1-\beta_2) \\
m_t &= m_{t-1} \cdot \beta_1 + g_t/\sqrt{v_t} \\
\Delta_t &= \operatorname*{argmax}_{\|\Delta\|=1} \langle \Delta, m_t\beta_1 + g_t/\sqrt{v_t} \rangle \\
O_t &= \beta_{\text{offset}} O_{t-1} + (1-\beta_{\text{offset}})\eta_t \Delta_t \\
W_{t+1} &= W_t - (O_t + \Delta_t)
\end{aligned}
$$

where $\|\cdot\|$ is a norm obtained by taking the max of all the appropriate norms for each tensor. This makes solving for Δ_t decompose tensor-by-tensor.

5 Experiments

Experiment setup We tested the performance of our algorithm (which we called "mango") on the GPT2 124M model and the pile dataset [18]. The model is trained with batch size 128 and 2000 iterations for a total of 0.25B tokens, and a fixed random seed is applied across all runs. We use a fixed learning rate schedule, linear decay with warmup of 200 steps, across all runs for a fair comparison. We expect that replacing with the trapezoid schedule could further improve the performance, but such improvement is orthogonal. We highlight important results in this section and provide a full wandb report here. A JAX implementation of our algorithm can be found here.

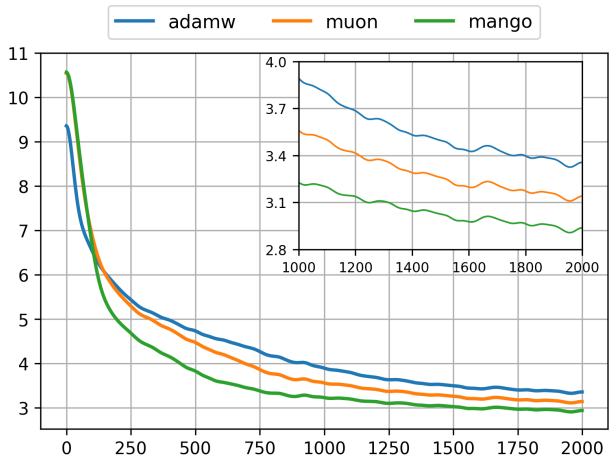


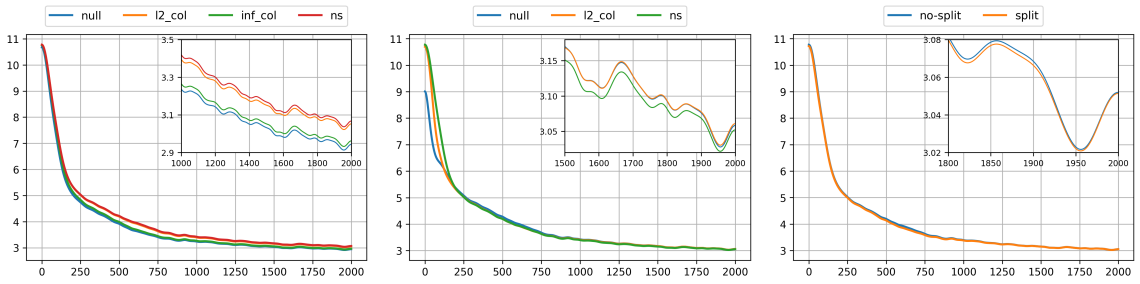
Figure 1: Performance of different optimizers, adamw, muon, and mango, with optimal hyperparameters. We fine-tuned the learning rates via grid search and set peak learning rate to 1e-3 for adamw, 0.01 for muon with 3e-4 for the adamw counterpart, and 0.01 for mango for all layers. For other hyperparameters, we followed the default values of adamw and muon; tuning mango is explained later.

Testing normalization of different layers One focus of the experiment is to verify the aforementioned theoretical hypothesis of selecting norms in Section 3. We partitioned the model into the following subgroups: embedding matrices (weights in token and position embedding), head matrix, attention weights and bias (as used for computing attention score), other 2d weight matrices (e.g., in expanding or reducing linear layers in attention blocks), other 1d weight vectors (weights in LayerNorm), other 1d bias vectors. We tested the following norms for each layer (see Figure 2):

- For 1d vectors, we tested normalizations with ℓ_2-norm $\|\cdot\|_2$ and ℓ_∞-norm $\|\cdot\|_\infty$.

- For 2d matrices, we tested normalization with operator norm $\|\cdot\|_{\text{op}}$ and column-wise vector norms. Operator normalization is approximated by the Newton-Schulz algorithm, as employed by muon. Column-wise vector normalization is implement as $M[i] \mapsto \frac{M[i]}{\|M[i]\|}$ where $M[i]$ denotes the i-th column of matrix M and $\|\cdot\|$ is either ℓ_2 or ℓ_∞ norm.
- For the attention weights and bias, we tested "head-wise" normalization. By default, the QKV matrices in the multi-head attention blocks are implemented as one single matrix of size $\mathbb{R}^{3nd\times D}$ where n denotes the number of heads and d denotes dimension of each head. In other words, each head corresponds to a $\mathbb{R}^{d\times D}$ sub-matrix. As an intuitive alternative, we applied operator norm on the sub-matrices instead of the entire matrix, and similarly applied ℓ_2 norm on the $\mathbb{R}^d$ sub-vectors of the $\mathbb{R}^{3nd}$ bias vector.

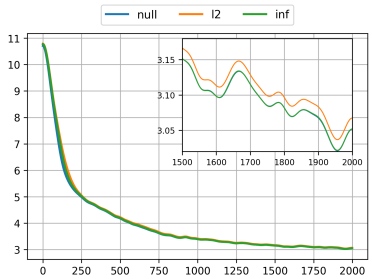
As a remark, the optimal norm for each tensor is as follows: no normalization for the embedding weights (while column-wise $\|\cdot\|_\infty$ is almost equally good), $\|\cdot\|_{\text{op}}$ for other weight matrices, $\|\cdot\|_2$ for bias vectors, and no normalization for weight vectors ($\|\cdot\|_\infty$ is almost equally good). Additionally, using head-wise norms for attention weights and bias



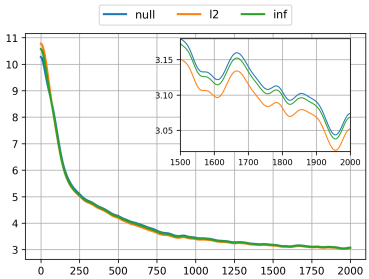
(a) Embedding layers

(b) Head layer

(c) Attention layers



(d) Weight vectors



(e) Bias vectors

Figure 2: Testing different norms for different layers. "null" denotes no normalization, "l2" and "inf" denote $\|\cdot\|_2, \|\cdot\|_\infty$ on vectors, "l2_col" and "inf_col" denote column-wise norms on matrices, "ns" denotes $\|\cdot\|_{\text{op}}$ on matrices implemented by Newton-Schulz, and finally "split" and "no-split" denote whether head-wise normalization is applied to attention weights and bias. In each subplot, we fixed the norms for all layers except for one. By default, we set column-wise ℓ_2-norm for embedding, operator norm for all other matrices, ℓ_∞-norm for weight vectors, and ℓ_2-norm for all bias vectors.

Gradient preconditioning and offset momentum We also tested different values of β_2 and β_{offset} (see Figure 3). Notably, gradient preconditioning $g_t/\sqrt{v_t}$ provides a notable improvement (over 0.2 in the final loss), while offset momentum provides a less significant improvement of roughly 0.01. Combining all these components, our algorithm achieves the performance shown in Figure 1.

Acknowledgements

We thank Harsh Mehta for helpful discussions.

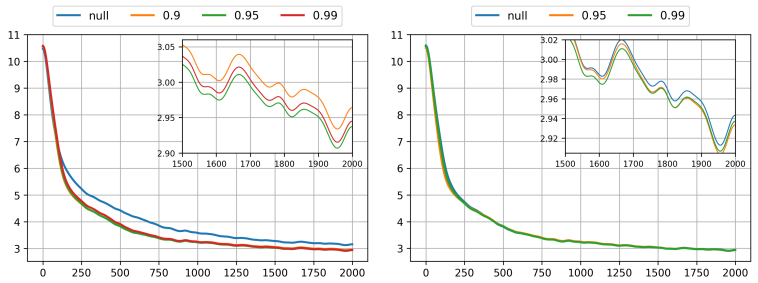


Figure 3: Tuning different β_2 with $\beta_{\text{offset}} = 0.99$ (left) and tuning different β_{offset} with $\beta_2 = 0.95$ (right). We used the optimal norms discussed earlier.

References

[1] Jeremy Bernstein and Laker Newhouse. "Modular Duality in Deep Learning". In: *arXiv preprint arXiv:2410.21265* (2024).

[2] Tim Large, Yang Liu, Minyoung Huh, Hyojin Bahng, Phillip Isola, and Jeremy Bernstein. "Scalable Optimization in the Modular Norm". In: *arXiv preprint arXiv:2405.14813* (2024).

[3] Liu Ziyin, Zhikang T. Wang, and Masahito Ueda. *LaProp: Separating Momentum and Adaptivity in Adam*. 2021. arXiv: `2002.04839 [cs.LG]`.

[4] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[5] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. *Muon: An optimizer for hidden layers in neural networks*. 2024. URL: `https://web.archive.org/web/20250122060345/https://kellerjordan.github.io/posts/muon/`.

[6] Yang You, Igor Gitman, and Boris Ginsburg. "Large batch training of convolutional networks". In: *arXiv preprint arXiv:1708.03888* (2017).

[7] B.T. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[8] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Conference on Learning Theory (COLT)*. 2010, pp. 257–269.

[9] H. Brendan McMahan and Matthew Streeter. "Adaptive Bound Optimization for Online Convex Optimization". In: *Proceedings of the 23rd Annual Conference on Learning Theory (COLT)*. 2010, pp. 244–256.

[10] Zachary Nado, Justin M Gilmer, Christopher J Shallue, Rohan Anil, and George E Dahl. "A large batch optimizer reality check: Traditional, generic optimizers suffice across batch sizes". In: *arXiv preprint arXiv:2102.06356* (2021).

[11] Vineet Gupta, Tomer Koren, and Yoram Singer. "Shampoo: Preconditioned stochastic tensor optimization". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1842–1850.

[12] Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. "Scalable second order optimization for deep learning". In: *arXiv preprint arXiv:2002.09018* (2020).

[13] MLCommons. "Announcing the results of the inaugural AlgoPerf: Training Algorithms benchmark competition". In: *MLCommons website* (2024). Accessed: 2024-8-6. URL: `https://mlcommons.org/2024/08/mlc-algoperf-benchmark-competition/`.

[14] Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. "Soap: Improving and stabilizing shampoo using adam". In: *arXiv preprint arXiv:2409.11321* (2024).

[15] Ashok Cutkosky and Harsh Mehta. "High-probability Bounds for Non-Convex Stochastic Optimization with Heavy Tails". In: *Proceedings of the 35st International Conference on Neural Information Processing Systems*. 2021, pp. 4883–4895.

[16] Yurii Nesterov. "A method for solving the convex programming problem with convergence rate O (1/k2)". In: *Dokl akad nauk Sssr*. Vol. 269. 1983, p. 543.

[17] Ashok Cutkosky and Francesco Orabona. "Black-Box Reductions for Parameter-free Online Learning in Banach Spaces". In: *Conference On Learning Theory*. 2018, pp. 1493–1529.

[18] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. "The Pile: An 800GB Dataset of Diverse Text for Language Modeling". In: *arXiv preprint arXiv:2101.00027* (2020).