Neon: Nuclear Norm to Beat Muon

Alexey Kravatskiy
kravtskii.aiu@phystech.edu

Ivan Kozyrev
kozyrev.in@phystech.edu

Nikolay Kozlov
kozlov.na@phystech.edu

Alexander Vinogradov
vinogradov.am@phystech.edu

April 27, 2025

In this paper, we develop a new algorithm for optimization of functions of weight matrices, which are typical for training large language models. Changing spectral norm, which was used to derive Muon, to nuclear norm, we pose a new optimization problem for an update matrix, solution of which defines a novel algorithm we name Neon. To make it feasible, we use Lanczos algorithm to find a required step. After providing theoretical guarantees of Neon convergence, we compare performances of Neon, Muon, and Adam on training multilayer perceptron, convolutional neural network and NanoGPT.

1 Idea

The goal of the project is to make variations on Muon to speed it up. Recently, authors of [1] have proposed to look at different optimizers as the solution of the optimization problem for an update. This approach can be utilized to derive Muon [2], a novel algorithm for fast training of neural networks. Instead of using spectral norm in the problem, we use nuclear norm to produce a new problem.

1.1 Problem (Project description)

In this subsection, we provide a more detailed description of our idea and formulate it as a mathematical problem. The authors of [1] suggest obtaining the update step as a solution to the optimization problem:

$$\langle g, \delta w \rangle + \lambda \|\delta w\|^2 \to \min_{\delta w}, \tag{1}$$

where w is the weight vector, g is a gradient-like vector (e.g., obtained via momentum SGD), and $\|\cdot\|$ represents a certain norm. Many popular optimizers, such as Adam (with exponential moving average disabled) and vanilla SGD, can be cast within this framework [1].

In large language models, most weights are structured as matrices, which offers additional opportunities for optimization. Let W be the weight matrix of a linear layer, and G be a gradient-like matrix. Then, the update step δW can be obtained as a solution to the optimization problem:

$$\langle G, \delta W \rangle + \lambda \|\delta W\|^2 \to \min_{\delta W}, \tag{2}$$

where $\|\cdot\|$ denotes a certain matrix norm. By setting this norm to the RMS-to-RMS norm (a scaled version of the spectral norm), we recover the Muon optimizer [3, 1] with an update step defined by:

$$\delta W = -\frac{1}{\lambda}\sqrt{\frac{n}{m}} U V^T, \tag{3}$$

where m is the input dimension of the layer, n is the output dimension, and U and V are obtained from the singular value decomposition of the gradient matrix $G = U\Sigma V$.

Motivated by the recent achievements of the Muon optimizer (e.g., [4]), we consider alternative choices of norms, specifically the kernel norm $\|\cdot\|_*$ and a custom $F*$ norm, given by $\|X\|_{F*}^2 = (\|X\|_F + \|X\|_*)/2$, where $\|\cdot\|_F$ denotes the Frobenius norm.

Using the kernel norm in (2) leads to a rank-one update of the weight matrices:

$$\delta W = -\frac{1}{\lambda} u_1 \sigma_1 v_1^T \,, \tag{4}$$

where σ_1 is the largest singular value, and u_1 and v_1 are the corresponding singular vectors. We expect one iteration of this method to be significantly faster than one iteration of Muon.

Another choice is the $F*$ norm. With this choice, (2) yields

$$\delta W = -\frac{1}{\lambda} U D V^T \tag{5}$$

with $D = \mathrm{diag}(d_i)$, where $d_i = [\sigma_i - \tau]_+$ and τ is given by:

$$\sum_{i=1}^{n} [\sigma_i - \tau]_+ = \tau \,. \tag{6}$$

We anticipate that the method with this update step will perform well with large batch sizes.

In this article we show how one can quickly compute weight updates defined by (4) or (5) (for now we need more tests to accept one variant or both of them). Then we finilize the method by adding momentum and test their performance against those of Muon at training multilayer perceptron and transformer. The results will be fast algorithm, which we will convert into a new optimizer classes for PyTorch, as was done with Muon.

2 Outcomes

The main results of the project are expected to be:

1. Deriving of Neon and argumentation of the numerical methods which will be used in the algorithm
2. Code implementation of Neon as PyTorch optimizer class
3. Numerical experiments comparing performance of training with Neon, Muon, SVD and Adam on several architectures: MLP, CNN, and NanoGPT.
4. Participation in CIFAR-10 challenge with Neon (if the alrothim turns out to be competitive)
5. Article describing all the results. If they are satisfactory, we intend to be in time to apply for NeurIPS 2025. It is a challenge, but I think we must give it a try.

3 Literature review

Our review primarily focuses on the Muon and Shampoo optimizers, as our algorithm extends the ideas used to derive these methods. We highlight the advantages and disadvantages of these approaches, the unique effects they introduce, and compare them to Neon.

3.1 Muon optimizer

In the previous section, we described the theoretical foundation behind the weight update step in the Muon optimizer, but we did not discuss how to obtain the required matrices in practice. The update step is defined by (3), which requires UV^T with U and V^T from the singular value decomposition of the gradient-like matrix G. A naive solution would involve computing the SVD of G and constructing the required expression. However, the developers of the Muon optimizer introduced a workaround using Newton-Schulz iterations [2]. The Newton-Schulz iterations from the original article [2] require 10 matrix-matrix multiplications to achieve the desired accuracy. The asymptotic complexity of such an operation is identical to that of SVD and equals $O(mn \min\{m, n\})$ for an $m \times n$ matrix. Nevertheless, matrix multiplication on modern GPUs can be performed much more efficiently.

The performance of Muon in training large language models was tested [4] against AdamW. The testing demonstrated excellent performance by Muon, which was approximately 2 times more efficient in terms of FLOPs required to reach a certain loss value. This is even more remarkable when considering the cost of one iteration: Muon requires an additional $O(mn \min\{m, n\})$ FLOPs per $m \times n$ matrix, while AdamW needs only $O(mn)$.

Another interesting discovery about the Muon optimizer is that it accelerates grokking [5]. In the test problem, Muon achieved grokking significantly faster than AdamW in terms of passed epochs, with a mean grokking epoch of 102.89 for Muon and 153.09 for AdamW. The authors suggest that this may be due to the fact that Muon stimulates broader exploration by orthogonalizing the gradient matrix, thus avoiding memorization.

Recently, theoretical guarantees for Muon convergence have been derived [6]. In particular, in the L-smooth convex case, it achieves $O(1/T^{\frac{1}{2}})$ (with full gradient) and $O(1/T^{\frac{1}{4}})$ (with stochastic gradient) bounds on the Frobenius norm of the gradient or the mathematical expectation of the gradient norm, respectively, where T is the number of iterations.

3.2 Shampoo optimizer

Another optimizer that exploits the matrix (and even tensor) structure of weights in neural networks is the Shampoo optimizer. We avoid a detailed description of this method here and refer the reader to the original article [7], but we outline the key properties of Shampoo and its relation to the Muon optimizer.

The Shampoo optimizer uses left and right preconditioning for the gradient-like matrix, leveling its spectrum. The preconditioners are computed from the exponentially averaged gradients, and their computation requires $O(n^3 + m^3)$ per $m \times n$ matrix. This exponential averaging is a key feature that provides several distinct interpretations for the preconditioners. They can be viewed as an approximation of the Gauss-Newton component of the Hessian or the first step of the power iteration algorithm for computing the optimal Kronecker product approximation [8]. With exponential averaging turned off, the update step of Shampoo can be simplified and becomes identical to that of Muon [2].

Convergence analysis is presented in the original article [7]. Shampoo achieves $O(1/T^{\frac{1}{2}})$ convergence for the loss function value in the L-smooth convex case, where T is the number of iterations.

3.3 Synthesis and Neon's position

Recent developments in optimization techniques show that utilizing the matrix structure of weights in neural networks can be very beneficial. Optimizers following this path converge faster in terms of iterations or epochs and often even FLOPs, but have a high iteration cost. Neon seeks to decrease the iteration cost while preserving fast convergence.

While the uniqie advantages and effects introduced by Neon are yet to be discovered, we can already say that our new optimizer introduces additional overhead of $O(mn)$ FLOPS on average per $m \times n$ weight matrix, which is much better then $O(mn \min\{m, n\})$ for Muon optimizer and $O(n^3 + m^3)$ for Shampoo.

3.4 Optimization Strategies for Efficient Large Language Model Training

The unprecedented scale of modern Large Language Models (LLMs) has pushed traditional optimizers like AdamW [9] to their limits in terms of computational efficiency and convergence speed [4, 10]. This challenge has catalyzed research into more sophisticated optimization approaches that can maintain or improve performance while reducing training costs.

The Muon optimizer has emerged as a promising alternative based on matrix orthogonalization principles. Scaling Muon to billion-parameter LLMs required two key adaptations: the integration of L2 weight decay for stability and the implementation of per-parameter update scaling to handle diverse parameter distributions efficiently . Empirical evaluations demonstrate that Muon can match or exceed AdamW's model quality while requiring only about 52 of the training FLOPs. The successful training of the Moonlight model series, including a 16B-parameter Mixture-of-Experts model, validates Muon's practicality for production-scale applications [4].

Building on this foundation, hybrid approaches like COSMOS [10] further enhance efficiency by combining optimization techniques based on gradient structure. COSMOS applies computationally intensive updates to a low-dimensional "leading eigensubspace" while using memory-efficient methods like Muon for the remaining parameters. This approach maintains convergence benefits while substantially reducing memory requirements. For distributed training environments, optimizers like Dion [11] specifically target communication efficiency by minimizing data exchange between workers through distributed orthonormalization techniques.

4 Quality metrics

1. The derivation is theoretically solid

2. The numerical procedure used to compute a step is grounded and has estimated time overhead (say, in FLOPS)
3. The code with Neon trains MLP and CNN (and NanoGPT, but it's a bonus) less than 3 times slower than Adam
4. Instruction of setting the parameters of the algorithm are presented and justified
5. The announced article has full structure (Abstract, Introduction, Theory, Experiments, Conclusion, Appendix)
6. If results are positive, it is written with NeurIPS template.

5 Preliminary plan

Week April 28 - May 4

- For Alexey: solve how to tune the algorithms for MLP and CNN, try formulating theory (and an appropriate model of the problem) why Muon and Neon are so successful, and create the drafts of the proofs. Register at NeurIPS site.
- For Ivan: write the theory for an update from the algebra point of view (as for an article)
- For Nikolay: write the theory for computing an update, and implement the method, if required
- For Alexander: reproduce results of Jordan on NanoGPT and ResNet (CIFAR-10), learn to train both models with Neon.

Week May 5 - May 11

- For Alexey: finalize the proofs. Verify them via small experiments on MLP and CNN. Write with Alexander Experiments for the article.
- For Ivan: join Nikolay to finalize algebra part of the article. Estimate FLOPS, memory and other overheads (produce O bounds)
- For Nikolay: write a draft of the poster (before May 6), and work with Ivan
- For Alexander: agressively test algorithms, prove that Neon outperforms competitors and prepare the results for the article.
- For everybody: write and edit the article
- May 11: submit an abstract to NeurIPS.
- May 12-14: the article is being polished.
- May 15: the article must be sent.

6 Prototyping phase report

1. Update rule is derived, see idea
2. Update rule methods are tested: power iteration vs Lanczos (see playground)
3. Recorded the distribution of singular values of gradients during NanoGPT training (see Firgures 1, 1).
4. NanoGPT is tested on Muon and Adam (add fig). The results are strange, because the methods do not converge at the expected rate (Muon is slower). Besides, Neon (rank-1 version) does not converge (see Figures 3 4)

   The pictures show the best results achieved so far. The experiments were conducted with two 4090 24GB GPUs for nanotgpt-large on the tiny stories dataset
5. Neon (rank-1 version), Muon, AdamW and SGD are compared on MLP and CNN (see Figures 5, 6, 7, and 8). All methods work correctly, but again there is the problem with which one is the fastest (for now, it's SGD).

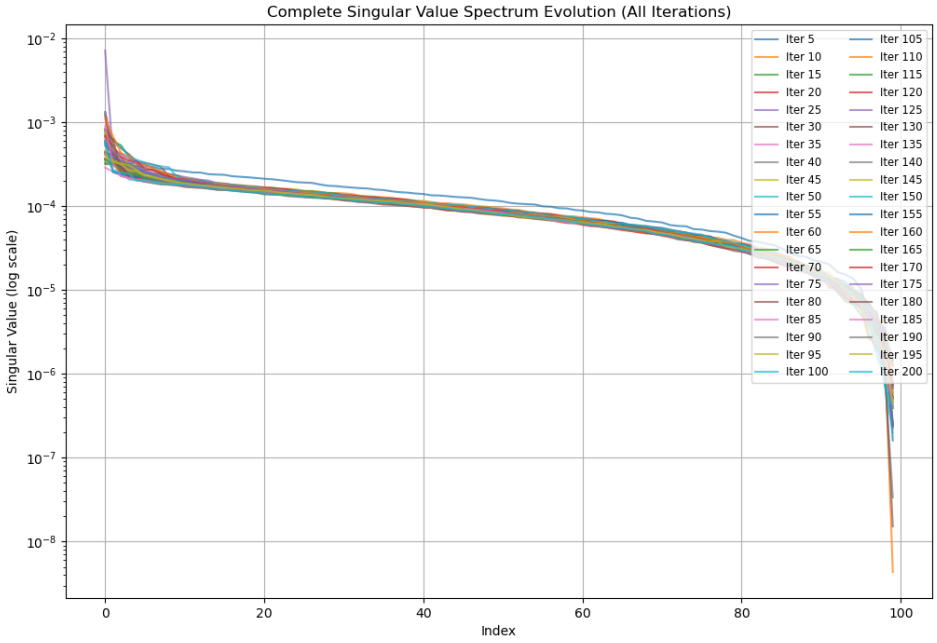

Figure 1: Singular values of 50257×1280 layer via 200 iterations

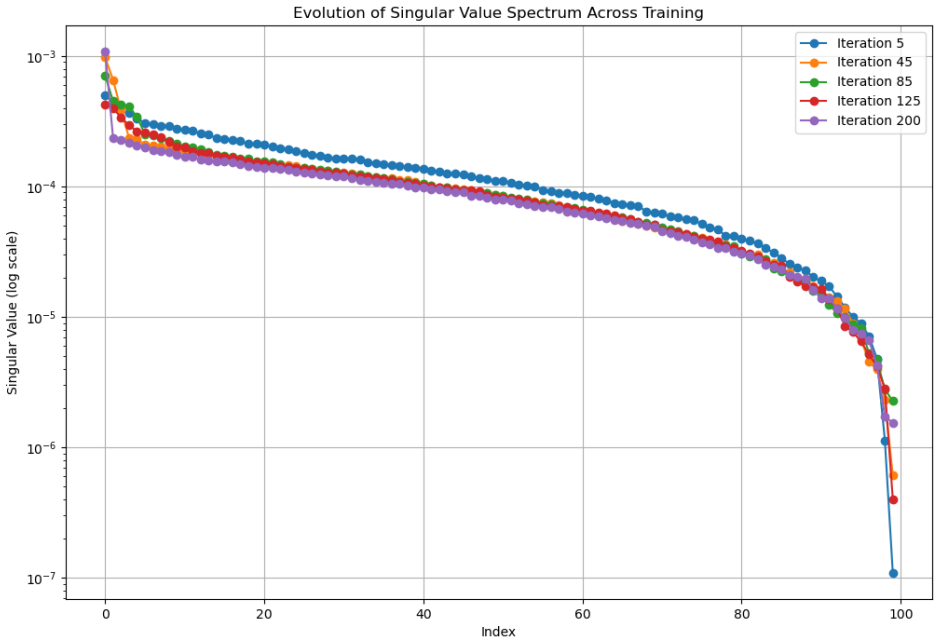

Figure 2: Singular values of of 50257×1280 layer for 5-th,45-th, 65-th,175-th and 200-th iteration

References

[1] Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. *arXiv preprint arXiv:2409.20325*, 2024.

[2] 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.

[3] Jeremy Bernstein. Deriving muon, 2025.

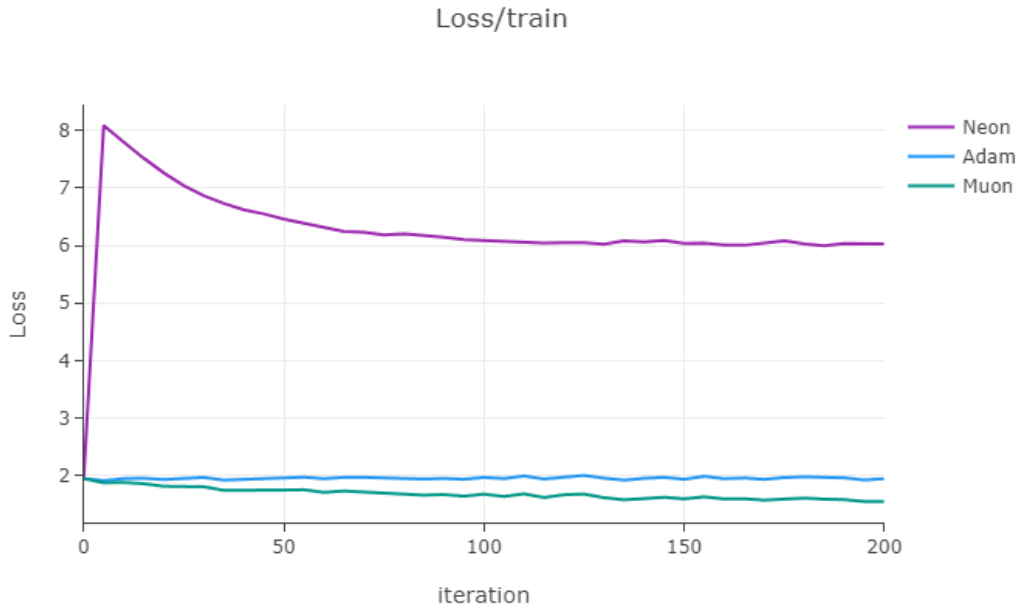

Figure 3: Train loss

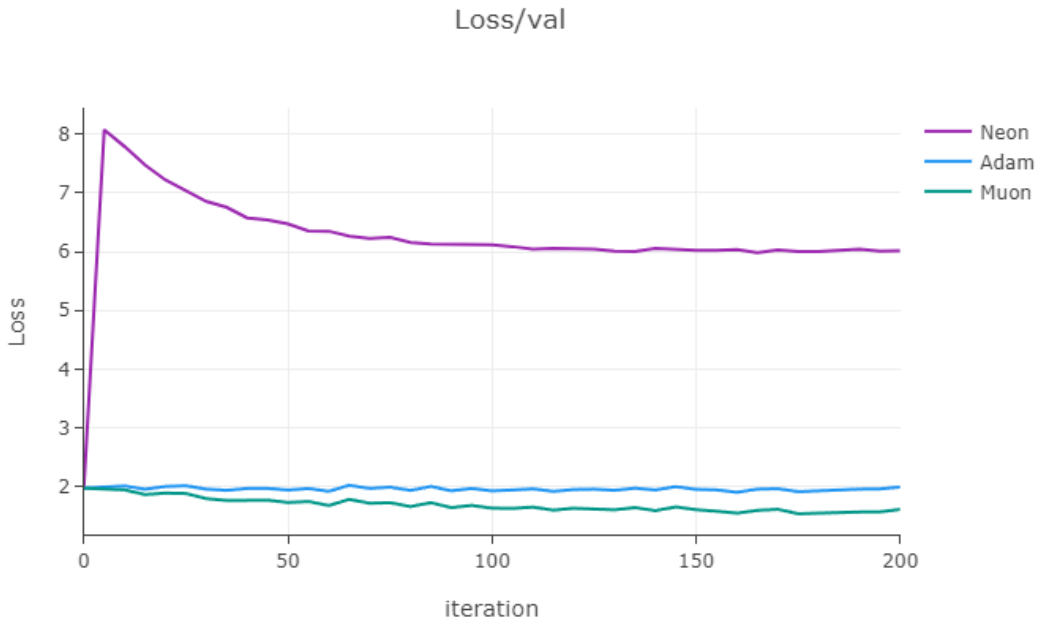

Figure 4: Validation loss

[4] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

[5] Amund Tveit, Bjørn Remseth, and Arve Skogvold. Muon optimizer accelerates grokking, 2025.

[6] Jiaxiang Li and Mingyi Hong. A note on the convergence of muon and further, 2025.

[7] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization, 2018.

[8] Depen Morwani, Itai Shapira, Nikhil Vyas, eran malach, Sham M. Kakade, and Lucas Janson. A new perspective on shampoo's preconditioner. In *The Thirteenth International Conference on Learning Representations*, 2025.

[9] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. *ArXiv*, abs/1711.05101, 2017.

[10] Weizhu Chen, Chen Liang, Tuo Zhao, Zixuan Zhang, Hao Kang, Liming Liu, Zichong Li, and Zhenghao Xu. Cosmos: A hybrid adaptive optimizer for memory-efficient training of llms, 2025.

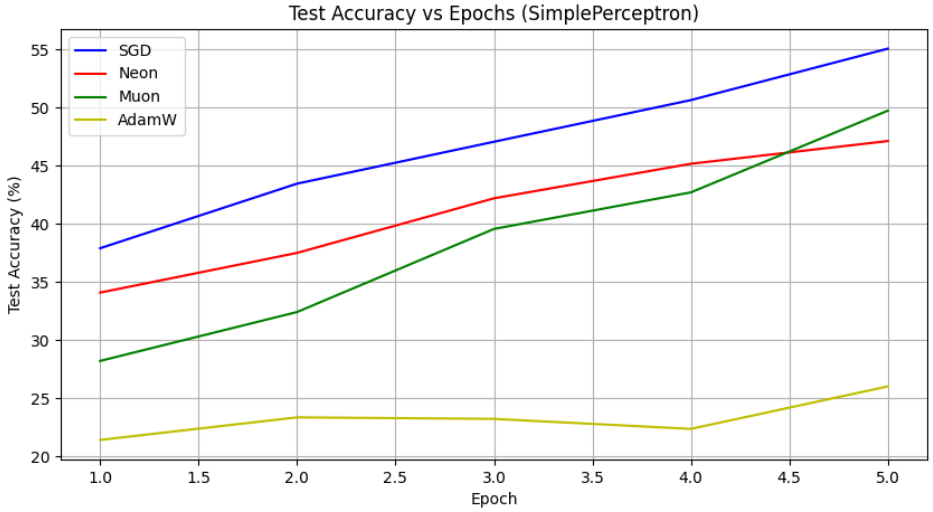

Figure 5: MLP: self.linear1 = nn.Linear(32*32*3, 512), self.linear2 = nn.Linear(512, 10), self.activ = nn.GELU()

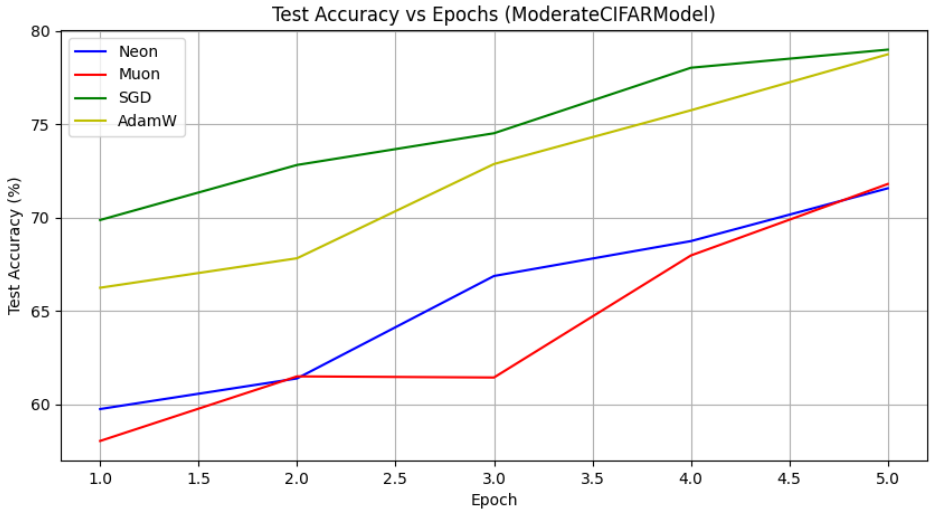

Figure 6: CNN: 2 convolutional blocks, 2 fully connected layers, activation + dropout

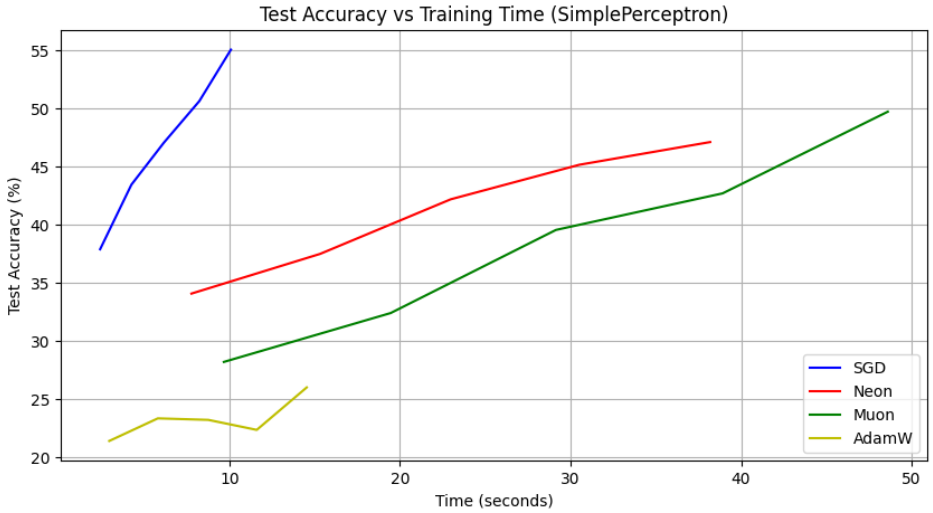

Figure 7: MLP: wallclock time measurements

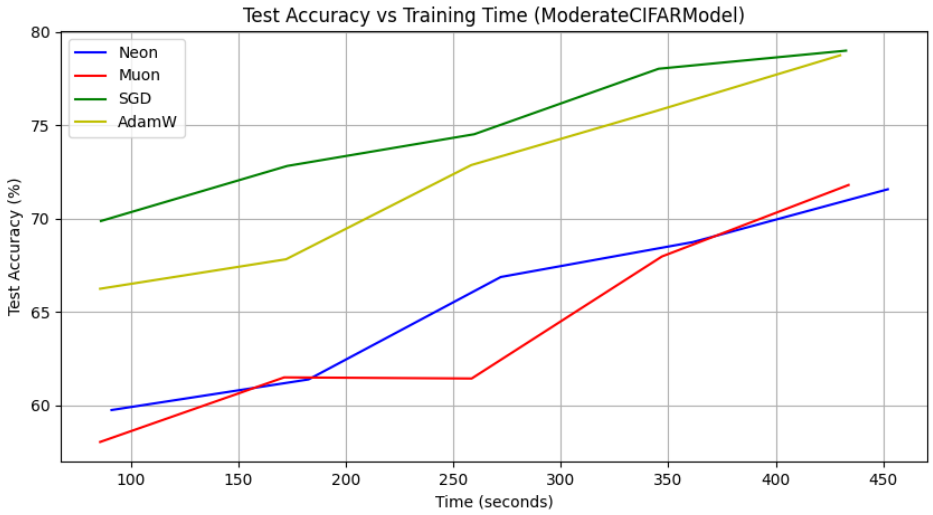

Figure 8: CNN: wallclock time measurements

[11] Kwangjun Ahn and Byron Xu. Dion: A communication-efficient optimizer for large models, 2025.