Deconstructing What Makes a Good Optimizer for Language Models

Rosie Zhao[*] ROSIEZHAO@G.HARVARD.EDU
Harvard University

Depen Morwani[*] DMORWANI@G.HARVARD.EDU
Harvard University

David Brandfonbrener[*] DAVID_BRANDFONBRENER@G.HARVARD.EDU
Kempner Institute at Harvard University

Nikhil Vyas[*] VYASNIKHIL96@GMAIL.COM
Harvard University

Sham Kakade SHAM@SEAS.HARVARD.EDU
Kempner Institute at Harvard University

Abstract

Training language models becomes increasingly expensive with scale, prompting numerous attempts to improve optimization efficiency. Despite these efforts, the Adam optimizer remains the most widely used, due to a prevailing view that it is the most effective approach. We aim to compare several optimization algorithms, including SGD, Adafactor, Adam, Lion, and Sophia in the context of autoregressive language modeling across a range of model sizes, hyperparameters, and architecture variants. Our findings indicate that, except for SGD, these algorithms all perform comparably both in their optimal performance and also in terms of how they fare across a wide range of hyperparameter choices. Our results suggest to practitioners that the choice of optimizer can be guided by practical considerations like memory constraints and ease of implementation, as no single algorithm emerged as a clear winner in terms of performance or stability to hyperparameter misspecification. Given our findings, we further dissect these approaches, examining two simplified versions of Adam: a) signed momentum (Signum) which we see recovers both the performance and hyperparameter stability of Adam and b) Adalayer, a layerwise variant of Adam which we introduce to study the impact on Adam's preconditioning for different layers of the network. Examining Adalayer leads us to the conclusion that, perhaps surprisingly, adaptivity on *both* the last layer and LayerNorm parameters in particular are necessary for retaining performance and stability to learning rate.

1. Introduction

As language model architectures increase in scale, pretraining becomes more expensive. In response, numerous efforts have been made to design efficient optimizers to mitigate these costs, and yet Adam [17] remains the primary optimizer used for training language models. This persistent preference for Adam is rooted in an underlying belief that Adam generally outperforms alternative optimization algorithms. Although newly proposed optimizers run ablations to demonstrate superior performance to Adam for select architectures and tasks [7, 21], there is no consensus among the literature about the relative performance of these optimizers. In fact, to the best of our knowledge,

[*] Equal contribution, randomized author ordering. Correspondence to rosiezhao@g.harvard.edu.

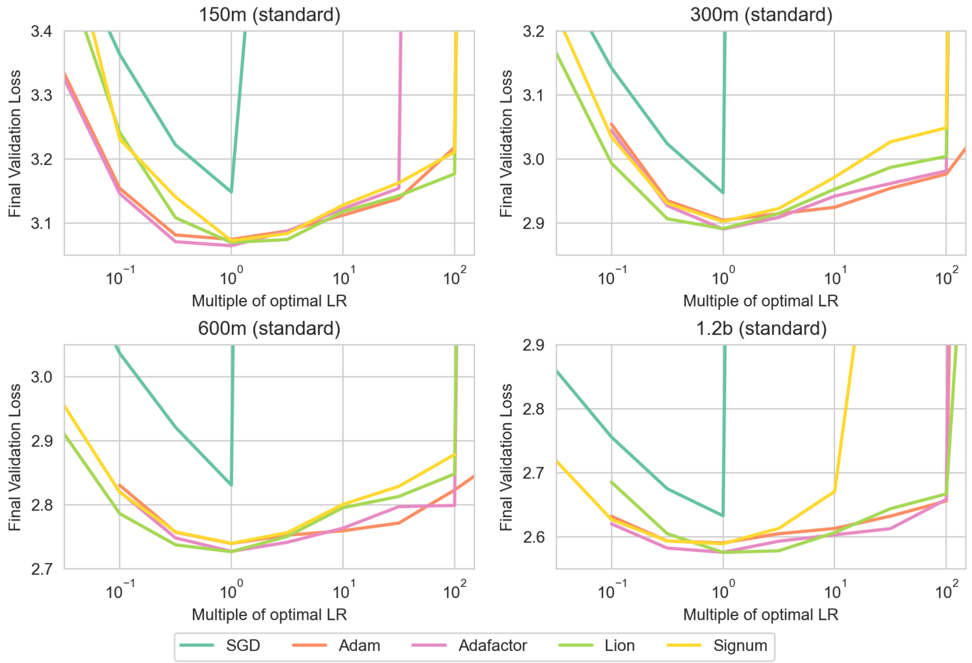

Figure 1: Final validation loss when training language models with 150m, 300m, 600m, and 1.2b parameters, sweeping across learning rates for five standard optimizers (SGD, Adam, Adafactor[1], Lion, and Signum). Plots have been shifted to align the optimal learning rates for each optimizer. Except for SGD, other optimizers seem comparable in their optimal performance and stability with respect to learning rate tuning.

[16] is the only work comparing these optimizers but in the context of masked language modeling and at a single model scale.

In this work, we perform a comprehensive sweep for training autoregressive language models across different optimizers, hyperparameters, architectures, and scale. Along with looking at optimal performance, we argue that due to the difficulty of hyperparameter tuning with increasing scale [36], the *stability* of performance with respect to hyperparameter choices is equally important. Prior work has explored the learning rate stability of Adam [35]. We extend this investigation to include the stability of multiple optimizers with respect to various hyperparameter choices. Surprisingly, we find that multiple optimizers introduced in the literature after Adam—such as Lion [7] and Adafactor (with momentum) [32, 37]—demonstrate robustness comparable to Adam and significantly superior to SGD. Figure 1 illustrates the remarkable similarity in performance and robustness of these optimizers across different learning rates and across multiple model scales (150m, 300m, 600m, and 1.2b parameters). This challenges the prevailing notion that Adam should be the default optimizer, where we see no single algorithm emerged as a clear winner in terms of performance or hyperparameter stability.

Following our initial ablations, we wish to identify the essential components of these optimizers that facilitate performance and stability. Thus, we conduct a series of investigations of simplified versions of these algorithms. We study signed momentum (Signum), a special case of Lion. Prior

[1] Our implementation of Adafactor adds back momentum as described in Appendix B.

works have also studied its similarities to Adam [3]. We find that Signum also recovers the stability and performance exhibited by Adam. This finding aligns with recent work [19], suggesting that the primary distinction between SGD and Adam is driven by Adam's resemblance to signSGD.

To further understand the role of preconditioning on various network parameters, we study Adalayer, which performs preconditioning on a *per-layer* basis. We empirically demonstrate that this variant nearly recovers the stability and performance of the other optimizers in previous ablations. Through empirical studies of Adalayer and its variants, we show that while adapting the parameters of the last layer and LayerNorm parameters in a transformer is necessary to achieve stability and performance, we can actually train the remaining parameters (a vast majority of the network) with SGD. To summarize, our main contributions are as follows:

- We empirically study the stability to hyperparameters of various optimization algorithms including SGD, Adam, Lion and Adafactor, showing that with the exception of SGD, these optimizers are comparable in terms of both performance and hyperparameter stability. This holds across multiple scales (150m, 300m, 600m, and 1.2b) and across two transformer architecture variants (Section 2).
- We study a coarser variant of Adam called Adalayer, that does per-layer preconditioning and recovers much of the stability and performance exhibited by Adam (Section 3.1). Through an empirical study of Adalayer and its variants, we establish that adaptivity is only necessary for the last layer and LayerNorm parameters, while the remaining parameters can be trained with SGD (Section 3.2).

2. Comparing Optimizers Across Hyperparameters, Architectures and Scale

2.1. Methodology

To conduct our experiments, we start with hyperparameters recommended by previous work (e.g., $\beta_1 = 0.9$). We initially perform a learning rate sweep to identify the optimal learning rate. After determining the optimal learning rate for each algorithm, we conduct one-dimensional sweeps for each of the other hyperparameters. A limitation of this methodology is the potential neglect of ”2D” interactions between hyperparameters. This is an important direction for future work, but beyond the computational budget of this project. For example, some parameters like batch size and learning rate indeed are likely to exhibit 2D interactions [27, 31]. However, we argue that the 1D sweeps provide a tractable methodology that gives us useful signal about the hyperparameter stability of a variety of algorithms around the parameters that are common in practice.

2.2. Setup

We train language models on C4 tokenized with the T5 tokenizer [28] and report results in terms of validation loss. As we discussed in the introduction, we argue that it is best to evaluate algorithms both in terms of the loss achieved by the best hyperparameters (performance) as well as the robustness across values of the hyperparameters (stability). Full details of hyperparameters and our setup can be found in Appendix B. In the next section we first present the results of sweeps across learning rate. We sweep across five algorithms: Adam, Adafactor, Lion, Signum, and SGD. Further ablations of momentum, weight decay, warmup, β_2, and ϵ for the 150m standard model can be found in Appendix C.

2.3. Sweeping learning rates

First, we sweep over the most important hyperparameter: learning rate. Note, in all of these sweeps over learning rate we set $\beta_1 = 0.9$ for all algorithms except for SGD, where we set $\beta_1 = 0.98$. As we will see in the following subsection, SGD is more sensitive to the momentum hyperparameters and requires more momentum to be competitive with the other optimizers.

Main results for our standard architecture across three scales are presented in Figure 1. Note that the x-axis shifts the learning rates to align the optimal learning rates across algorithms. In terms of absolute learning rates, we sweep in multiples of $\sqrt{10}$ from 1e-4 to 1 for Adam and Adafactor, from 1e-5 to 1e-1 for Lion and Signum, and from 1e-3 to 10 for SGD.

The key takeaway is that not only do the algorithms achieve similar performance at the optimal learning rate, but the learning rate stability itself is similar across algorithms and scales. The one exception is SGD, which is worse both in terms of optimal performance and in terms of stability. In Appendix C.4, we perform additional experiments specifically investigating the close performance between Adam and Signum.

Takeaway: performance and stability to learning rate are comparable across the non-SGD algorithms that we tested.

3. Investigating the key factors for optimizer stability and performance

Ablations in the previous section revealed the striking similarity in performance and stability across multiple optimizers compared to Adam. Adam and its other variants are designed to have a high degree of adaptivity at a fine granularity (per-parameter learning rates) throughout the training process. This adaptivity is often credited with the stability and robust performance observed in these optimizers. However, a critical question arises: to what extent is this adaptivity needed for different parameters of the network? By identifying the necessity of adaptivity for different network components to ensure both performance and stability, we aim to discern whether simpler optimizers like SGD can achieve similar benefits with minimal modifications. Since higher momentum can often play the same role as a better preconditioner and to have all algorithms on an equal footing, we will fix $\beta_1 = 0.9$ for all optimizers in this section.

The main optimizer we study in this section is a "layer-wise" variant of Adam, which we coin as 'Adalayer'. We use Adalayer for our investigations because it lends a greater ease of understanding compared to full-fledged Adam in identifying parts of the network which may be particularly critical for optimizer performance and stability. Note that this layerwise variant is a special case of a previously known optimizer called Blockwise Adaptive Gradient with Momentum (BAGM) [40].

3.1. Adalayer

To study the behavior of adaptive optimizers like Adam, we begin with describing a layer-wise version of Adam which we refer to as Adalayer. Adam, Adafactor and Adalayer all (approximately) store the diagonal second moment matrix, but with coarser and coarser granularity; for a layer of dimension $m \times n$, Adam explicitly maintains the second moment matrix using mn parameters in the shape of a matrix. Adafactor stores row and column averages of the second moment matrix which serve as a rank-1 approximation to the second moment matrix. Finally, Adalayer stores a single scalar which is the average of the second moment matrix. We will later consider a generalization of Adalayer where instead of averaging second moment over a layer we will average it over a

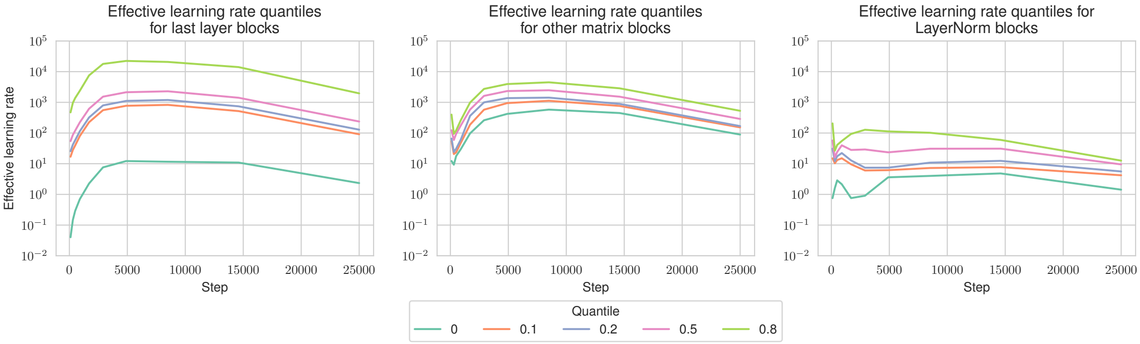

Figure 2: Quantiles of effective learning rates ($\eta_t/(\sqrt{v_t^l}+\epsilon)$ for each layer l) for the last layer blocks (**Left**), the LayerNorm blocks (**Right**), and the other matrix blocks (**Middle**) for a 150m model trained using Adalayer*. Unlike the other matrix blocks and LayerNorm parameters, the effective learning rates across logits vary across multiple orders of magnitude, providing evidence for the need to precondition them separately.

"block" of parameters which can be a subset of a layer. We note that similar algorithms have been studied before [1, 10] but we choose to study this variant since it is a direct analogue of Adam and Adafactor.

A simplified version of Adalayer optimizer is given in Algorithm 1; other details such as bias correction are kept same as that for Adam. In Appendix D, we provide more details about our Adalayer implementation. Specifically, Adalayer when naively applied for each layer is neither performant nor stable to learning rate (Figure 9); however, if we additionally treat the set of weights in the last layer feeding into each logit as its own block, this recovers most of the performance and stability of Adam (see the dotted blue lines in Figure 3). We henceforth refer to Adalayer with this correction as **Adalayer***. To study how Adalayer* preconditions the network, we plot effective learning rates used for different logits by Adalayer* in Figure 2 (**Left**). Here, the effective learning rate for a layer l in the network is $\lambda_t/(\sqrt{v_t^l}+\epsilon)$. We find that Adalayer* indeed uses vastly different learning rates for different logits, supporting our hypothesis that preconditioning weight in different logits separately is important for performance and stability.

Algorithm 1 Adalayer

```
Parameters: Learning rate $\eta$, exponential decay rates for the moment estimates $\beta_1, \beta_2$, number of steps $T$, $\epsilon$
while $t \leq T$ do
    for each layer $l$ with $p$ parameters do
        $g_t^l \leftarrow \nabla_l L(w_t)$
        $v_t^l \leftarrow \beta_2 \cdot v_{t-1}^l + (1-\beta_2) \cdot p^{-1/2} \cdot \|g_t^l\|_2^2$
        $m_t^l \leftarrow \beta_1 \cdot m_{t-1}^l + (1-\beta_1) g_t^l$
        $w_{t+1}^l \leftarrow w_t^l - \eta \cdot \frac{m_t^l}{\sqrt{v_t^l}+\epsilon}$
    end
end
```

3.2. Both the last layer and LayerNorm parameters need adaptivity

The results using Adalayer* in the previous section suggest that all layers except the last layer only need a iteration-dependent scalar correction to their learning rate. We now ask a stronger question: do we need these scales at all? Or can we train the remaining layers with SGD? This hypothesis is supported by looking at Figure 2 (middle) where we observe that the learning rates for different matrix layers (except the last layer) assigned by Adalayer* are remarkably similar.

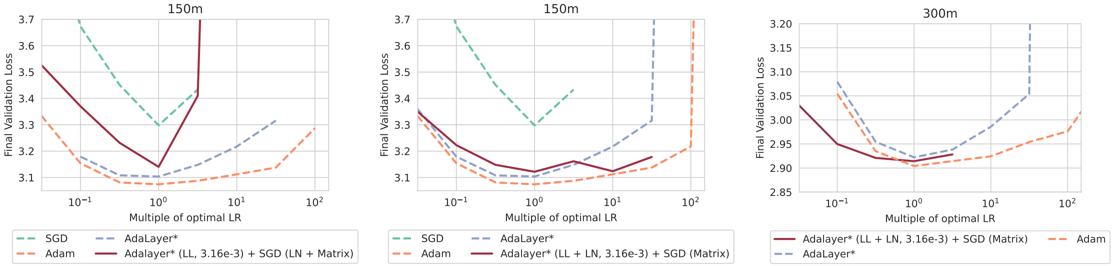

Figure 3: (**Left**): Training the last layer using Adalayer* with a fixed learning rate of $3.16e-3$ and other LayerNorm and matrix blocks using SGD achieves better performance than SGD, but does not recover stability. (**Middle**): Training both the last layer and LayerNorm blocks using Adalayer* and the other matrix blocks using SGD nearly recovers or exceeds performance of Adalayer*, and achieves stability across learning rates. (**Right**): Training both the last layer and LayerNorm blocks using Adalayer* and the other matrix blocks using SGD for a 300m model. This outperforms Adalayer* and is comparable to Adam's performance. Dotted lines are baselines from optimizers previously given in Sections 2 and 3.1.

To test this, we train the last layer with Adalayer* (fixing a learning rate of $3.16e-3$) and the rest of the layers with SGD, both with $\beta_1 = 0.9$. In Figure 3 (**Left**) we show the results while sweeping over SGD learning rates from 0.1 to 3160. While this improves upon the performance of SGD, we do not recover stability of the Adalayer* and Adam baselines. We trace this instability to *LayerNorm blocks*: Figure 2 (**Right**) shows that the effective learning rates for the LayerNorm blocks are much smaller, which suggests that they may destabilize at higher SGD learning rates. To ameliorate this, in Figure 3 (**Middle** and **Right**) we add LayerNorm parameters to those being trained with Adalayer* and find that this is sufficient to recover both performance and stability of Adalayer*. For the larger 300m model, we find that this even exceeds the performance of Adalayer*. In Figure 10 in the Appendix, we see this trend continue to hold for 600m parameter models.

We conduct additional experiments investigating these 'hybrid' variants of SGD in Appendix E, where we provide a series of ablations supporting the evidence that Adalayer* on specifically both the last layer and LayerNorm parameters are key for establishing performance and stability. Also note that a caveat of the above results is that we have introduced an additional hyperparameter— SGD learning rate, which we are sweeping over— while keeping the Adalayer* learning rate fixed. While decoupling the learning rates here is needed (due to SGD's performant learning rates being orders of magnitude higher than that of Adalayer*), this may be responsible for the observed stability. To address this, in Appendix F we train the network with Adalayer* but we *stop updating the second moment estimates* for all layers *except the last layer and LayerNorm blocks* after initialization. We show that even in this setting, we can completely recover the performance and stability of Adalayer* on the whole network.

4. Discussion and Limitations

After a comprehensive comparison of a variety of optimizers for language modeling, we have found that many optimizers seem to be roughly equivalent both in terms of optimal performance and hyperparameter stability. Diving deeper, we have shown that the treatment of the last layer and LayerNorm parameters is crucial for realizing the benefits of adaptive optimizers. Of course, there are several limitations to our study including the fact that due to computational constraints we only ablate a few architecture decisions, that we only consider one dimensional hyperparameter sweeps, we fix batch size, and that we limit our study to autoregressive language modeling with a single dataset. Despite these limitations, we believe that the study sheds new light on the fundamentals of optimization for language modeling and suggests that optimizer choice may not be the optimal point of intervention for increasing efficiency.

References

[1] Naman Agarwal, Rohan Anil, Elad Hazan, Tomer Koren, and Cyril Zhang. Disentangling adaptive gradient methods from learning rates. *CoRR*, abs/2002.11803, 2020. URL https://arxiv.org/abs/2002.11803.

[2] Kwangjun Ahn, Xiang Cheng, Minhak Song, Chulhee Yun, Ali Jadbabaie, and Suvrit Sra. Linear attention is (maybe) all you need (to understand transformer optimization). In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=0uI5415ry7.

[3] Lukas Balles and Philipp Hennig. Dissecting adam: The sign, magnitude and variance of stochastic gradients, 2018. URL https://openreview.net/forum?id=S1EwLkW0W.

[4] Lukas Balles, Fabian Pedregosa, and Nicolas Le Roux. The geometry of sign gradient descent, 2020.

[5] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 560–569. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/bernstein18a.html.

[6] Jeremy Bernstein, Jiawei Zhao, Kamyar Azizzadenesheli, and Anima Anandkumar. signSGD with majority vote is communication efficient and fault tolerant. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJxhijAcY7.

[7] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V Le. Symbolic discovery of optimization algorithms. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=ne6zeqLFCZ.

[8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113, 2023.

[9] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pages 7480–7512. PMLR, 2023.

[10] Boris Ginsburg, Patrice Castonguay, Oleksii Hrinchuk, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Huyen Nguyen, Yang Zhang, and Jonathan M Cohen. Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. *arXiv preprint arXiv:1905.11286*, 2019.

[11] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

[12] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1837–1845. PMLR, 2018. URL http://proceedings.mlr.press/v80/gupta18a.html.

[13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[14] Samy Jelassi, David Dobre, Arthur Mensch, Yuanzhi Li, and Gauthier Gidel. Dissecting adaptive methods in gans, 2022.

[15] Kaiqi Jiang, Dhruv Malik, and Yuanzhi Li. How does adaptive optimization impact local neural network geometry? In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=gIG8LvTLuc.

[16] Jean Kaddour, Oscar Key, Piotr Nawrot, Pasquale Minervini, and Matt J Kusner. No train no gain: Revisiting efficient training algorithms for transformer-based language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[18] Ananya Kumar, Ruoqi Shen, Sebastien Bubeck, and Suriya Gunasekar. How to fine-tune vision models with SGD. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZTssMmhC2X.

[19] Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=a65YK0cqH8g.

[20] Frederik Kunstner, Robin Yadav, Alan Milligan, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why adam outperforms gradient descent on language models. *CoRR*, 2024.

[21] Hong Liu, Zhiyuan Li, David Leo Wright Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3xHDeA8Noi.

[22] Yang Liu, Jeremy Bernstein, Markus Meister, and Yisong Yue. Learning by turning: Neural architecture aware optimisation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6748–6758. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/liu21c.html.

[23] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

[24] I.U.E. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Mathematics and its applications. Kluwer Academic Publishers, 2004. ISBN 9780004501444. URL https://books.google.com/books?id=klv2PwAACAAJ.

[25] Yan Pan and Yuanzhi Li. Toward understanding why adam converges faster than SGD for transformers. In *OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop)*, 2022. URL https://openreview.net/forum?id=Sf1NlV2r6PO.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[27] Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. *arXiv preprint arXiv:2406.19146*, 2024.

[28] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[29] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[30] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

[31] Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019. URL http://jmlr.org/papers/v20/18-789.html.

[32] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR, 2018. URL http://proceedings.mlr.press/v80/shazeer18a.html.

[33] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[34] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[35] Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie E Everett, Alexander A Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-Dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=d8w0pmvXbZ.

[36] Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 17084–17097. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/8df7c2e3c3c3be098ef7b382bd2c37ba-Paper.pdf.

[37] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 1204–1213. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01179. URL https://doi.org/10.1109/CVPR52688.2022.01179.

[38] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need adam: A hessian perspective, 2024.

[39] Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more, 2024. URL https://arxiv.org/abs/2406.16793.

[40] Shuai Zheng and James T. Kwok. Blockwise adaptivity: Faster training and better generalization in deep learning. *CoRR*, abs/1905.09899, 2019. URL http://arxiv.org/abs/1905.09899.

Appendix A. Related work

One closely related work to ours is Wortsman et al. [35], which explores the stability of Adam with respect to learning rate. We extend the comparison to other optimizers including SGD, Lion and Adafactor, as well as other hyperparameters including momentum and weight decay.

Optimizers: SGD [30] had been the workhorse optimizer for deep learning until 2015, when Adam [17] was introduced. Adam is a diagonal preconditioning algorithm that maintains a per-parameter learning rate. Over time, coarser variants of Adam have been proposed, which do not explicitly maintain a learning rate per parameter. Adafactor [32, 37] maintains a rank-1 approximation of the preconditioner matrix of Adam. Previous works have also explored Signum [5, 6] and have observed its benefits in terms of communication efficiency and fault tolerance. Other works have also explored the similarity of Adam with variants of Signum [3], and recently, a close variant of Signum, called Lion [7], was discovered using symbolic search over algorithms. Some other optimizers that have recently gained increasing attention from the community include Shampoo [12] and Sophia [21].

Adam and Signum: Many works have explored the relationship between Adam and variants of Signum [3, 4, 19] and empirically demonstrated that Signum (or its close variants) generally performs comparably to Adam. Balles et al. [4] also argued that signSGD generally performs better when the Hessian is close to diagonal, however, it is unclear if this holds for practical settings. Kunstner et al. [19] recently demonstrated that Adam and a close variant of Signum exhibit similar performance on a variety of datasets including WikiText-2 [23] and SQuAD [29]. However, in contrast with our work, all of these are restricted to the setting of vision or masked language modeling, and generally do not sweep over multiple hyperparameters.

Layerwise or blockwise Adam: We study Adalayer, a layerwise version of Adam. This is a special case of the BAGM optimizer [40], specifically BAGM B.1. Similar algorithms have also been studied by previous works [1, 10, 22, 39]. In particular, concurrent to our work, Zhang et al. [39] propose an algorithm termed Adam-mini, which closely tracks a modified version of Adalayer (called Adalayer*), and demonstrate comparable performance to AdamW. Note that, in our work, Adalayer* is introduced to understand the role played by preconditioning in Adam, and we do not specifically focus on the final performance. Zhang et al. [38] empirically study the Hessian spectrum of transformers at initialization and find it to be more heterogeneous across layers as compared to ResNets. They argue that this heterogeneity is evidence towards the importance of Adam in training transformers. In contrast our results (Section 3.2) show that Adam's preconditioning is particularly important for the last layer and LayerNorm parameters to achieve performance and learning rate stability.

Other related works: For vision transformers, in the fine-tuning phase, Kumar et al. [18] show that using SGD with frozen embedding parameters leads to competitive performance with Adam. Jelassi et al. [14] explore the similarity between Adam and normalized gradient descent [24] and show that normalized gradient descent on GANs does not suffer from mode collapse, while SGD does. Jiang et al. [15] empirically demonstrate that Adam steers the parameter trajectory towards better-conditioned regions than SGD. Pan and Li [25] also show that the parameter trajectory of Adam exhibits much higher directional smoothness than that of SGD. Ahn et al. [2] show that the performance gap between Adam and SGD exacerbates with depth of the network. In a similar vein to us, Kunstner et al. [20] show that Adam is less sensitive than gradient descent to class-imbalance present in language tasks; we provide further evidence for the importance of preconditioning the last layer, as well as the LayerNorm parameters.

Appendix B. Setup and Architecture Details

Algorithms. We use the standard Pytorch implementation of AdamW [26], the timm implementation of SGDW [34], and the OLMo implementation of Lion [11]. Following [37] we implement ourselves a modified version of Adafactor which maintains the factored estimates of second moments but **has momentum** i.e. it is equivalent to Adam with factored second moment estimates. Since Signum is equivalent to Lion with $\beta_1 = \beta_2$ we reuse the OLMo implementation of Lion [11] for it. We conducted experiments with the Sophia optimizer [21] in Appendix G. However, since it does not outperform Signum (which can be achieved by setting $\rho = 0$ in Sophia), we did not include it in other plots.

Models. We start from the OLMo codebase [11] and train decoder-only transformer models of three sizes: 150m, 300m, and 600m, where the parameter count refers to non-embedding parameters. The models have widths of 1024, 1024, and 1408 and depths of 12, 24, 24. The MLP hidden dimension is 4x of the width. The activation function is GeLU [13]. We use RoPE positional encodings [33]. Attention heads are always dimension 64. We use PyTorch default LayerNorm. Following previous work [35] we do not learn biases for the linear layers or LayerNorms. We train in mixed precision with bfloat16.

Training variants. We note that Wortsman et al. [35] observe that QK LayerNorm [9] and z-loss [8] can have substantial effects on the stability of model training. As such, we consider two variants in our experiments: **standard** which refers to a model with QK LayerNorms and z-loss with coefficient 1e-4, and **no QK norm or z-loss** which refers to the same model without the QK norm layers or the z-loss.

Token counts. For all models, we use a batch size of 256 and sequence length of 512 (as in Wortsman et al. [35]). We default to training models for the approximately "chinchilla optimal" number of tokens that is $\approx$20 times the number of parameters. Explicitly, this means for the 150m models we train for 25k steps or $\approx$3.3b tokens. The 300m models are trained for 50k steps, the 600m models are trained for 100k steps and the 150m-long models are also trained for 100k steps.

Other hyperparameters. We default to using 0 weight decay. We default to using a learning rate schedule with 10% of the training steps for warmup and then cosine decay with a minimum that is 10% of the maximum learning rate. We default to $\beta_2 = 0.95$ and $\epsilon = 1\text{e-}15$ following Wortsman et al. [35]. These parameters are ablated in Appendix C.3.

Appendix C. Additional Hyperparameter Sweeps

In Section 2, we reported our main learning rate sweeps across architectures, optimizers, and scale. Here, we report sweeps across other hyperparameters (i.e. momentum, β_2, warmup, ε, etc.).

Optimizer	Optimal Learning Rate
Adam	3.16e-3 (150m), 1e-3 (300m), 1e-3 (600m), 1e-3 (1.2b)
Adafactor	3.16e-3 (150m), 1e-3 (300m), 1e-3 (600m), 1e-3 (1.2b)
Lion	3.16e-4 (150m), 3.16e-4 (300m), 3.16e-4 (600m), 1e-4 (1.2b)
Signum	3.16e-4 (150m), 3.16e-4 (300m), 3.16e-4 (600m), 3.16e-4 (1.2b)

Table 1: Optimal Learning Rates for Various Optimizers

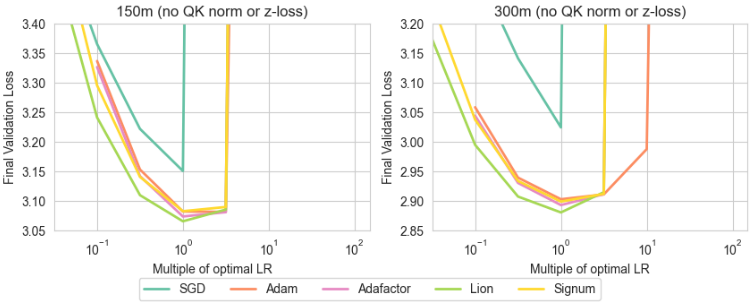

Figure 4: Sweeping learning rate without QK norm or z-loss for (**Left**) the 150m model, and (**Right**) the 300m model. These models are less stable than the standard model, but the same general trend across algorithms hold here.

C.1. Additional Learning Rate Sweeps

For our ablations in Figure 1, we report on the optimal learning rate found for each optimizer in Table 1. In general, we find that the optimal learning rate for Adam and Adafactor are similar, with the optimal learning rate of Lion and Signum an order of magnitude smaller.

Further ablations for learning rate are presented in Figure 4 and Figure 5 illustrating performance for models with no QK norm or z-loss and 4x longer training time respectively. While we find that the architecture choices can clearly impact the amount of stability to learning rate, the cross-algorithm comparisons remain the same: Adafactor and Lion are competitive with Adam, while SGD is worse both in terms of performance and stability to learning rate. Similarly, training for longer can improve performance and stability to learning rate, but does not change the high-level cross-algorithm comparisons.

C.2. Sweeping momentum

Now we also sweep across momentum values (i.e. β_1)[2]. To do this sweep we fix the per-algorithm learning rate to be the optimal learning rate from the corresponding learning rate sweep.

Results are presented in Figure 6. We observe that across various settings, the robustness to β_1 is similar across the non-SGD algorithms when we stay in the range of momentums between 0.8 and 0.98. However, for high β_1 Lion is better and low β_1 Adam and Adafactor are better. Again we observe SGD being very sensitive to momentum.

[2] Note that in Lion, both β_1 and β_2 can be thought of as different types of "momentum" with β_1 being the "one-step" momentum and β_2 the "long-term" momentum. For consistency, we only sweep β_1 here and sweep β_2 in Section C.3.

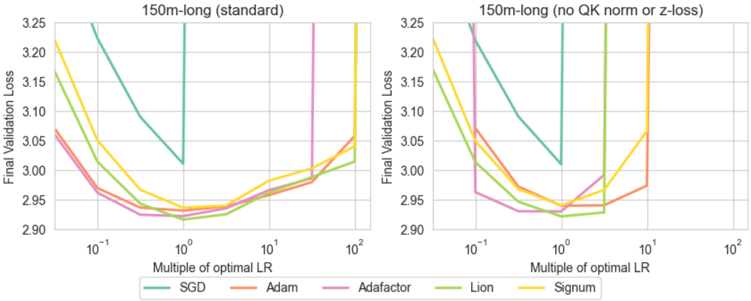

Figure 5: Sweeping learning rate on 150m models trained for 4x longer (100k steps) than in the base runs for (**Left**) the standard model, and (**Right**) the model without QK norm or z-loss. Compared to the shorter runs, these models achieve better performance and increased stability across learning rates.

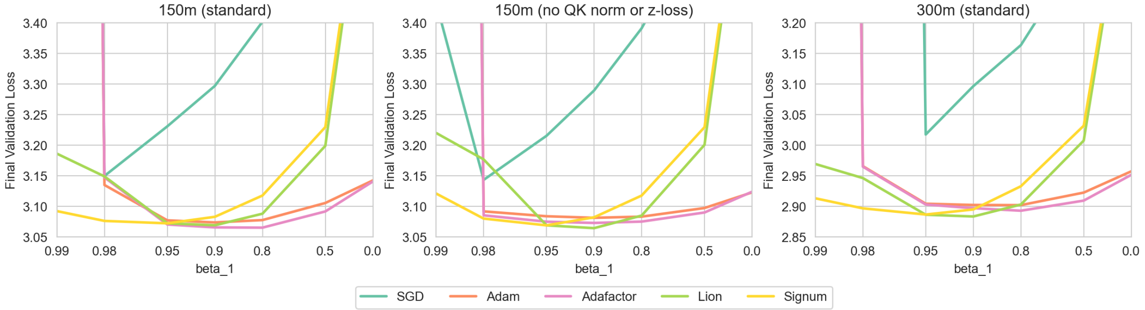

Figure 6: Sweeping momentum for fixed learning rate across three settings: (**Left**) 150m standard, (**Middle**) 150m with no QK norm or z-loss, (**Right**) 300m standard. Adam and Adafactor are similarly robust to β_1, while Lion and Signum are slightly more sensitive to low values and SGD is substantially more sensitive.

Takeaway: performance and stability to momentum are comparable across the non-SGD algorithms that we tested if we stay within the usual range of momentum values.

C.3. Additional hyperparameter sweeps

We also sweep over a variety of other hyperparameters in Figure 7 using the best per-algorithm learning rate and momentum. We observe that SGD is less stable with respect to weight decay and warmup length. And while it is possible to get small benefits from higher weight decay, longer warmup, and higher β_2 than our defaults, the algorithms are much more stable to these parameters than learning rate and momentum.

Takeaway: generally algorithms are more stable with respect to other hyperparameters and the possible gains in performance are relatively small compared to learning rate and momentum.

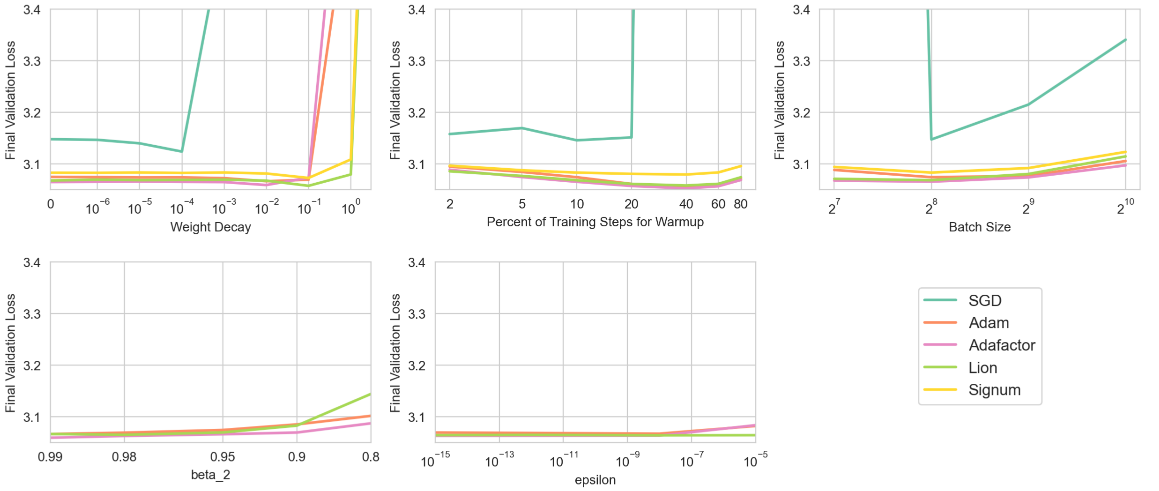

Figure 7: Sweeps over other hyperparameters. Top: weight decay, warmup duration, and batch size. Bottom: ϵ and β_2. We generally find little effect for the non-SGD algorithms, however there are parameters that differ from our defaults that can offer up to 0.02 improvements in perplexity.

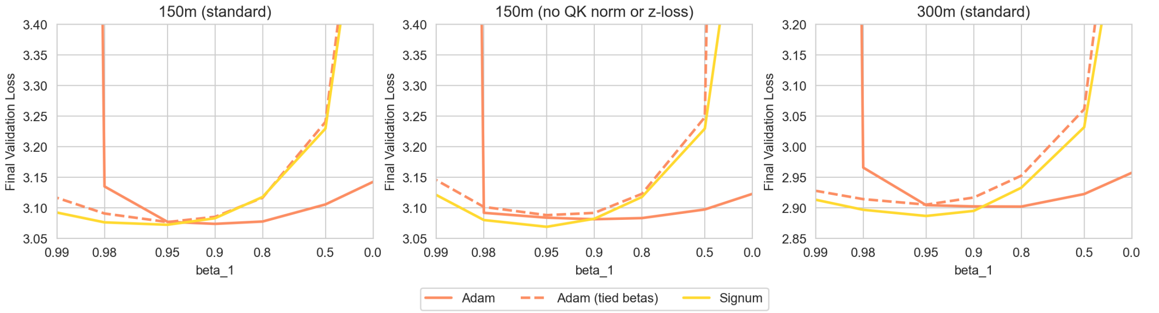

Figure 8: Sweeping momentum with $\beta_1 = \beta_2$ tied together for Adam (dashed) and compared to Signum and Adam with fixed $\beta_2 = 0.95$ (solid) across three settings: (**Left**) 150m standard, (**Middle**) 150m with no QK norm or z-loss, (**Right**) 300m standard. When $\beta_1 = \beta_2$, Adam behaves very similarly to Signum.

C.4. Signum recovers the performance and stability of Adam

In Figure 1 we observed that Adam and Signum have similar performance and stability for language modeling, even at scale. The following lemma from prior work[3] shows that Adam performs variance-adjusted sign gradient descent.

Lemma 1 ([3]) *Consider a parameter with a history of gradients $g_t, g_{t-1}, \ldots$. Let m be the random variable that is equal to $g_{t-\tau}$ with probability $(1-\beta_1)\beta_1^\tau$ and v be the random variable that is equal to $g_{t-\tau}$ with probability $(1-\beta_2)\beta_2^\tau$. The Adam update δ_{Adam} and the Signum update δ_{Signum} are related by*

$$\delta_{Adam} = \delta_{Signum} \cdot \frac{\mathbb{E}[m]}{\sqrt{\mathbb{E}[v^2]}}$$

If $\beta_1 = \beta_2$ then $m = v$ in Lemma 1 and hence the ratio of Adam and Signum updates is equal to the ratio of the mean and the square root of second moment of m. Intuitively, this holds because when $\beta_1 = \beta_2$, the first moment estimates of Signum and Adam, and second moment estimates of Adam, average the previous gradients with same coefficients ($(1-\beta)\beta^\tau$). This intuitively suggests that when $\beta_1 = \beta_2$, Adam and Signum may behave similarly. This motivates the conjecture that the main benefit of Adam over Signum is the fact that in Adam, β_2 can be varied independently of β_1. In Figure 1 we have $\beta_2 = 0.95$ and $\beta_1 = 0.9$ which are close, and as pointed out earlier, both optimizers have similar performance and stability.

We examine this hypothesis further in Figure 8 by varying β_1 and setting $\beta_2 = \beta_1$, and again find that Signum and Adam behave very similarly. However, we also note that when we vary β_1 for Adam while fixing β_2 we get more stability for β_1 as compared to Signum.

Takeaway: With $\beta_2 = \beta_1$ Adam and Signum behave similarly and the standard setting for training language models ($\beta_2 = 0.95, \beta_1 = 0.9$) is close to this.

Appendix D. Adalayer

As mentioned in Section 3, to investigate the role of preconditioning on language models for optimizers like Adam, we introduce the Adalayer optimizer for ease of analysis. In this section, we first establish the performance and stability of Adalayer as a reasonable proxy for Adam.

In Figure 9 we study the behavior of Adalayer across learning rates. To preserve the correspondence with Adam we fix other hyperparameters to be the same: $\beta_1 = 0.9$, $\beta_2 = 0.95$ and $\epsilon = 1e-15$. We find that Adalayer has better performance than SGD, but it performs worse than Adam and also lacks Adam's stability across learning rates. The major difference between Adam and Adalayer is the preconditioning done by Adam *within* a layer. Intuitively, this preconditioning will have large effects in layers where we expect different weights within a layer to have different gradient scales. The first candidate for such a layer is the *last layer*, since different tokens have widely different frequencies leading to different gradient scales. To test this hypothesis, we run a corrected version[3] of Adalayer where we treat the set of weights feeding into a logit as a separate block. We henceforth refer to Adalayer with this correction as **Adalayer***. This is plotted in Figure 9 and we observe that Adalayer* almost recovers the performance as well as a large fraction of the stability of Adam.

Appendix E. Additional experiments: SGD + adaptive variants (Adalayer*, Adafactor)

In this section, we report additional experiments involving training language models with SGD on a fraction of the models' parameters and an adaptive optimizer on the remaining parameters. Firstly, we show that our results from Section 3.2 hold even when training 600m parameter models with Adalayer* applied only on the last layer and LayerNorm parameters in Figure 10.

We also provide further ablations supporting our claim that the largest impact of the adaptivity of Adalayer* is concentrated on the last layer and LayerNorm parameters. Firstly, we train 150m models using Adalayer* on only the matrix parameters, while training the last layer and LayerNorm

[3] We note that this reasoning also applies to the first layer, but in our ablations applying this correction the first layer did not make a significant difference.

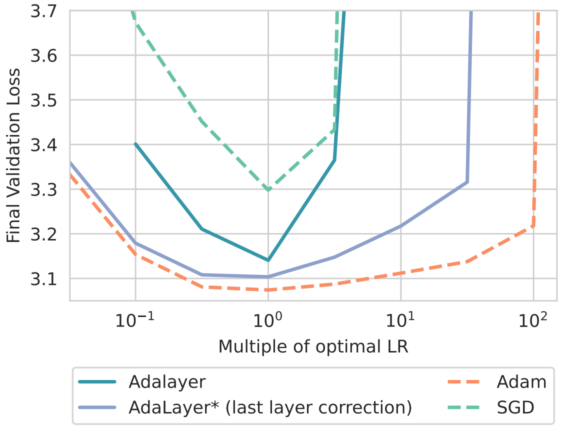

Figure 9: Modifying Adalayer with the last layer correction improves performance and stability across learning rates.

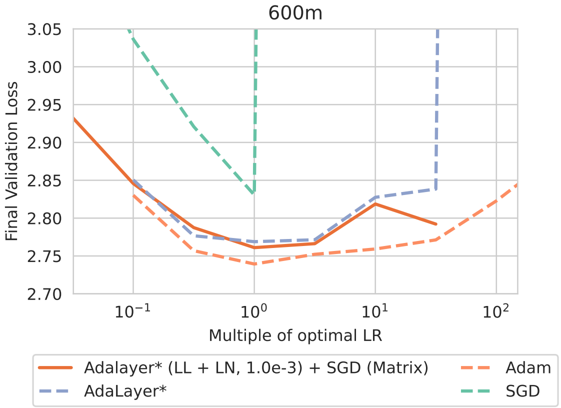

Figure 10: As in Section 3.2, we train 600m models with Adalayer* on the last layer and LayerNorm parameters, and train the remaining model parameters with SGD. We see that performance and stability continues to match that of Adalayer* even at this larger scale.

parameters with SGD. In Figure 11, we see performance improves relative to SGD but we see similar instability at larger learning rates.

Secondly, given that the effective learning rates of the LayerNorm blocks were observed to be small in Figure 2 (**Right**), it is reasonable to ask whether training the LayerNorm parameters is necessary at all; in Figure 12, we show results for training 150m and 300m models using Adalayer* only on the last layer, using SGD on all other matrix blocks, and turning off training for the LayerNorm parameters. This indeed yields greater stability in comparison to Figure 3 (**Left**) but does not fully recover the performance of Adalayer*, indicating that training LayerNorm parameters helps with performance, which seems more pronounced in the larger model.

We saw in Figure 3 (**Middle, Right**) that using Adalayer* on only the last layer and LayerNorm parameters sufficed to recover or exceed the performance of Adalayer*. In Figure 13, we report a learning rate sweep over the analogous experiment but using Adafactor on the last layer and

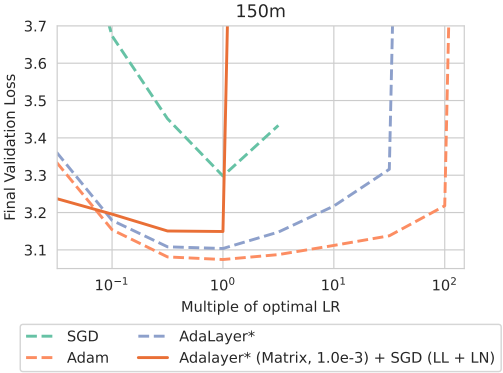

Figure 11: We train 150m models using Adalayer* on the matrix layers with a fixed learning rate of $1e{-}3$ and using SGD on the last layer and LayerNorm parameters. Compared to the results in Figure 3, we do not recover the same stability nor do we reach the optimal performance of Adalayer*.

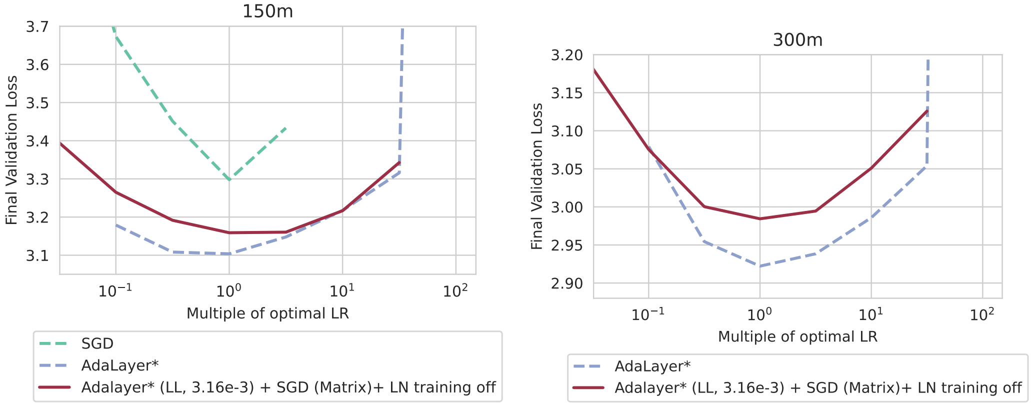

Figure 12: Training 150m (**Left**) and 300m (**Right**) models using Adalayer* on the last layer with a fixed learning rate of $3.16e-3$ and using SGD on other matrix blocks, while turning off the option to train LayerNorm parameters. We see that while the performance and stability is improved compared to SGD, it is still not as performant as Adalayer*. This indicates a degree of importance of training LayerNorm parameters for these models.

LayerNorm parameters with a fixed learning rate. For the 150m model, using a learning rate of $3.16e-3$ with Adafactor yielded better performance than Adafactor for low learning rates, and is comparable in terms of performance and stability. For the 300m model, the difference between Adafactor and our 'hybrid' optimizer is more distinct at higher learning rates for fixed Adafactor learning rate $1.0e-3$ and $3.16e-3$, but is comparable until the peak validation loss.

Appendix F. Additional experiments: freezing Adalayer learning rate ratios

As mentioned in Section 3.2, our experiments on the 'hybrid' SGD + Adalayer* optimizer has a potentially confounding factor that the Adalayer* learning rate for the last layer and LayerNorm parameters is fixed across SGD learning rate. In this section, we train all layers of the network with

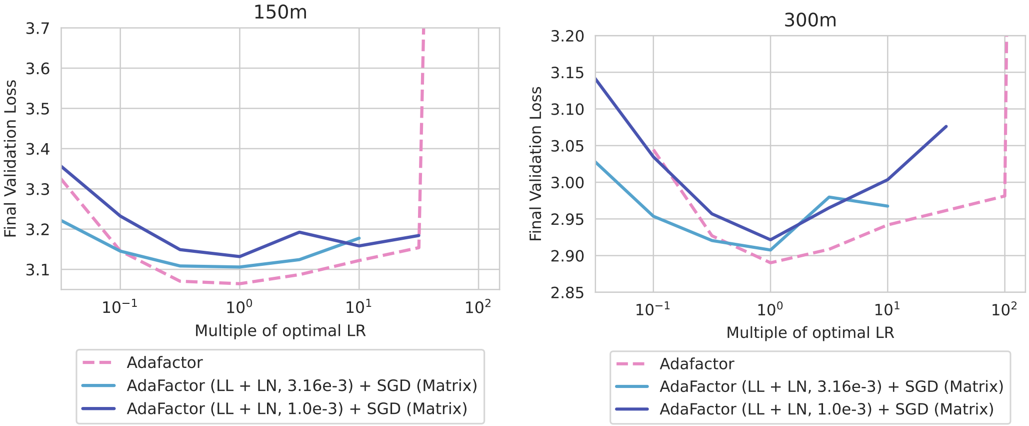

Figure 13: Training 150m (**Left**) and 300m (**Right**) models using Adafactor on the last layer with a fixed learning rate and using SGD on other matrix blocks. We see that performance and stability is comparable to Adafactor, but does not exceed it, particularly at higher learning rates.

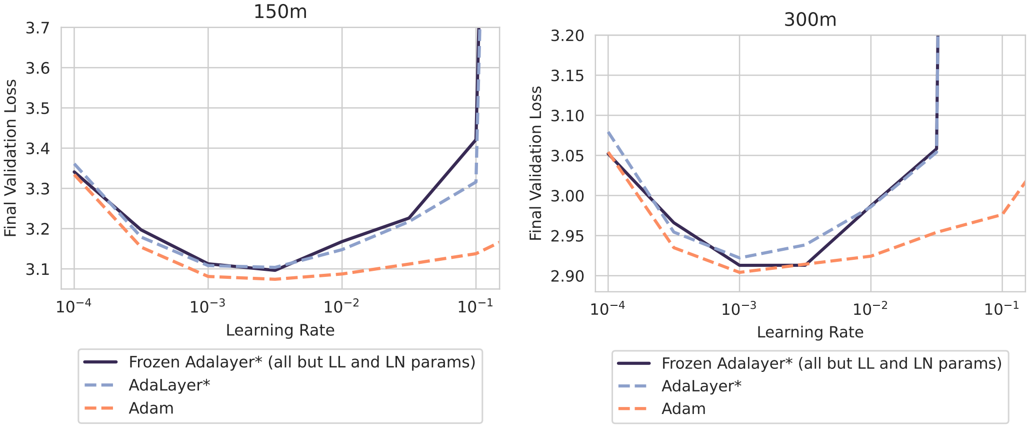

Figure 14: Training 150m (**Left**) and 300m (**Right**) models using fixed Adalayer* learning rate ratios from initialization, with the exception of last layer and LayerNorm parameters. This almost entirely matches the performance and stability of Adalayer* in the 150m model, and exceeds Adalayer*'s peak performance to be comparable with Adam.

Adalayer* after freezing the second moment estimates from initialization, *with the exception of the last layer and LayerNorm parameters*. This implies that these layers are effectively being trained by SGD with a fixed learning rate, though unlike the above results, these learning rates are different for different layers. We implement this by passing 1000 batches to initialized 150m and 300m models to obtain second moment estimates for all layers without letting the model take a gradient step, and then allowing the model to train as normal under the same settings as all of our ablations. As in our previous investigation, we fix other hyperparameters to be the same: $\beta_1 = 0.9$, $\beta_2 = 0.95$ and $\epsilon = 1e - 15$.

In Figure 14 we show the resulting learning rate sweep for freezing Adalayer* learning rate scales at initialization (with the exception of the last layer and LayerNorm). Surprisingly, we find for the 150m model that we can almost entirely recover the stability and performance of Adalayer*. For the 300m model we also match or exceed the performance of Adalayer*, and even nearly match the peak performance of Adam. Note again that this sweeps learning rate across all network parameters. This provides further evidence for the importance of adaptivity in the last layer and LayerNorm, where in contrast we could used fixed ratios from initialization for all other parameters to recover the performance and stability of Adalayer*.

We also report additional experiments exploring whether *both* last layer and LayerNorm adaptivity is truly needed for frozen Adalayer*. We show that this is indeed the case by conducting the same sweep for frozen Adalayer* while trying to also freeze the learning rate ratios for last layer or LayerNorm parameters as well. In Figure 15, we show that fixing initialized learning rate ratios for *all* layers does not reach peak performance of Adalayer*, nor does it exhibit stability. In Figure 16, we show that either continuing to update the LayerNorm parameters or the last layer parameters can achieve the peak performance of Adalayer* but is still unstable. Finally, we show results for turning off LayerNorm training while fixing learning rate ratios (with the exception of the last layer) in Figure 17. We conclude that it is necessary to maintain adaptivity for *both* the last layer and LayerNorm parameters, but understanding why the fixed ratios do not suffice would be an interesting question for future work.

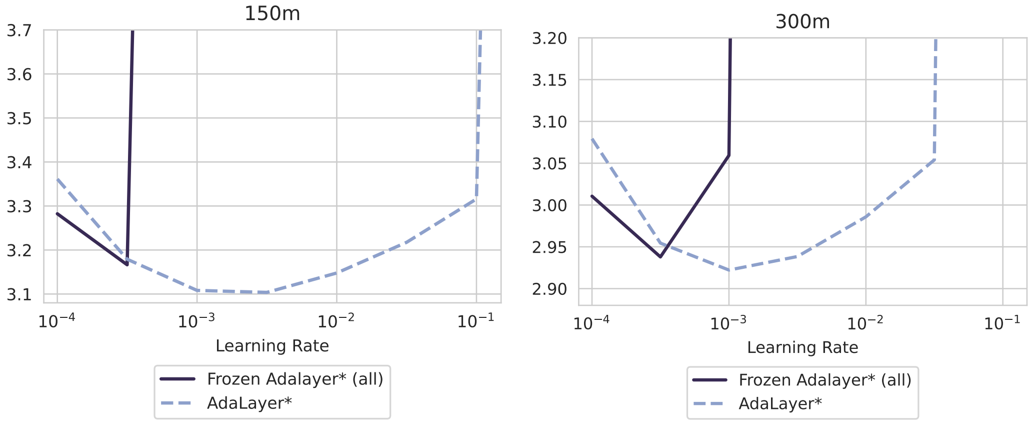

Figure 15: Training 150m (**Left**) and 300m (**Right**) models using fixed Adalayer* learning rate ratios from initialization for *all layers*. We observe this quickly diverges, achieving neither peak performance nor stability.

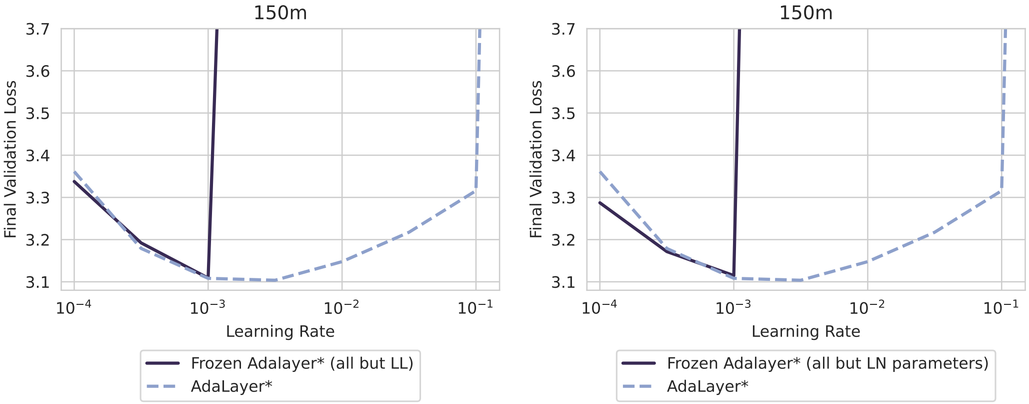

Figure 16: Training 150m models using fixed Adalayer* learning rate ratios from initialization while either excluding only the last layer (**Left**) or excluding only the LayerNorm parameters (**Right**). We observe both modifications reach peak performance but fails to be stable at higher learning rates.

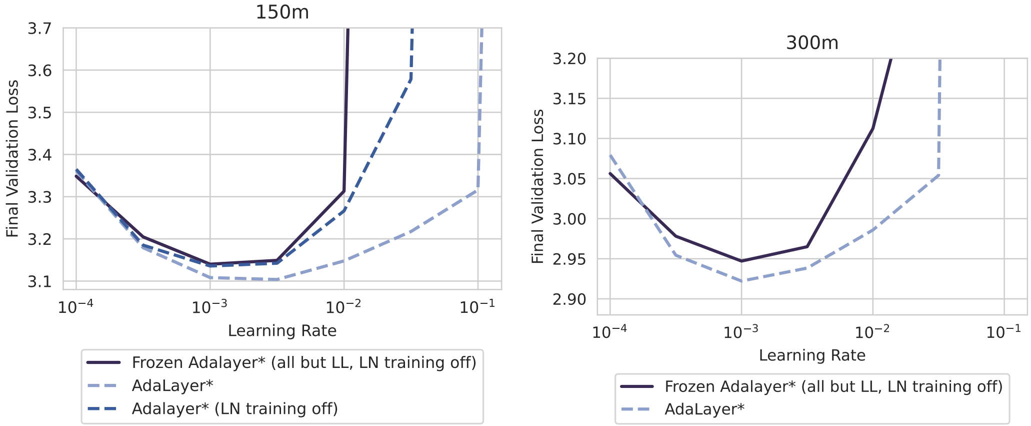

Figure 17: Training 150m (**Left**) and 300m (**Right**) models using fixed Adalayer* learning rate ratios from initialization, while letting the last layer continue to update, and turning LayerNorm training off. Stability across learning rates has improved but is less performant; for the 150m model we also plot the sweep for regular Adalayer* with LayerNorm training off, and we see that it is worse in performance compared to Adalayer* with LayerNorm training.

Appendix G. Sophia

In this section, we compare Sophia [21] to Signum. Note that Signum is a special case of Sophia, achieved by setting $\rho = 0$. We find that Sophia does not outperform Signum. No significant change in performance was observed when transferring the hyperparameters suggested by Liu et al. [21] (eg. β_1, β_2, ε, weight decay), nor when additionally scaling attention by the inverse of layer index which was used in the original Sophia implementation.

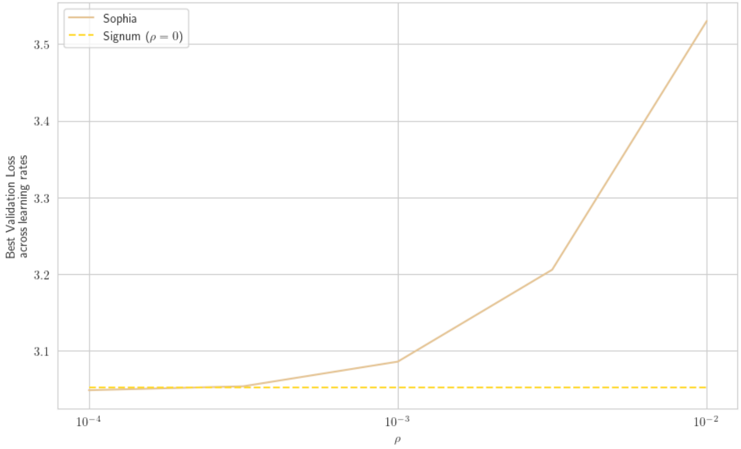

Figure 18: Comparing Sophia [21] and Signum for the 150M model in our default setup.