In Search of Adam's Secret Sauce

Antonio Orvieto ANTONIO@TUE.ELLIS.EU
ELLIS Institute Tübingen, MPI-IS, Tübingen AI Center, Germany

Robert M. Gower
CCM, Flatiron Institute, Simons Foundation, New York, US

Abstract

Understanding the remarkable efficacy of Adam when training transformer-based language models has become a central research topic within the optimization community. To gain deeper insights, several simplifications of Adam have been proposed, such as the signed gradient and signed momentum methods. In this work, we conduct an extensive empirical study — training over 1,300 language models across different data configurations and scales — comparing Adam to several known simplified variants. We find that signed momentum methods are faster than SGD, but consistently underperform relative to Adam, even after careful tuning of momentum, clipping setting and learning rates. However, our analysis reveals a compelling option that preserves near-optimal performance while allowing for new insightful reformulations: constraining the Adam momentum parameters to be equal, $\beta_1 = \beta_2$. Beyond robustness, this choice affords new theoretical insights, highlights the "secret sauce" on top of signed momentum, and grants a precise statistical interpretation: we show that Adam in this setting implements a natural online algorithm for estimating the mean and variance of gradients—one that arises from a mean-field Gaussian variational inference perspective.

1. Introduction

Despite a decade of research into efficient and performant adaptive optimizers for deep learning, the *de facto* choice for large-scale training today remains `Adam` [14], especially for training language models (LMs) [11, 19]. At the root of this choice is the peculiar geometry of optimization landscapes induced by the transformer architecture [24, 49], as well as the noisy/unbalanced nature of tokenized text data [17, 46]. In recent years, the surge of extremely large-scale and expensive-to-pretrain language models has further pushed the community to better understand `Adam`'s performance and to propose faster, efficient, and robust alternatives. Towards achieving this goal, contemporary studies [2, 16] have brought up a close similarity between the performance of `Adam` and `SignSGD` [3] with momentum. While such results are extremely valuable to forward our understanding, they are not precise enough : already at a scale of 160M parameters we found that extensive tuning of `SignSGD` with momentum (a.k.a. `Signum`), while closing 96% of

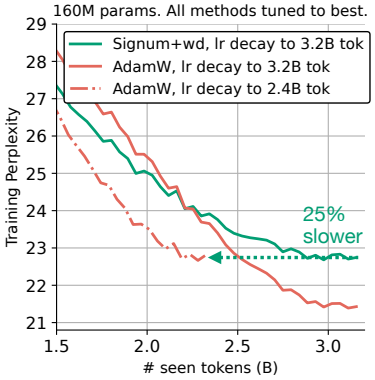

Figure 1: *Pretraining on SlimPajama with Chinchilla-optimal [12] scaling. Both momentum and learning rates for* `Signum` *are extensively tuned (§B). While* `Signum` *closes* 96% *of the perplexity gap between* `Adam` *and* `SGD` *with momentum (Table 1), still results in a* 25% *slowdown :* `Adam` *achieves the same performance with 3/4 of the budget.*

the perplexity gap between SGD and Adam, results in a 25% effective slowdown (Figure 1). The discussion above and the results in Figure 1 inspires us to further dissect – once again [1] – the mechanisms of Adam compared to those of simpler methods in language modeling.

All premilinaries on optimizers and related works can be found in §A.

	Adam	Signum	RMSprop	SGD+Cclip	SignSGD	SGD+Gclip	SGD
Val ppl.	**21.86**± **0.21**	23.23± 0.16	27.04± 0.34	33.40± 0.39	36.78± 0.57	37.76± 0.61	53.62± 5.14

Table 1: ***(Signum closes 96% of the perplexity gap between Adam and SGD)*** *Accuracy comparison of well-established optimization algorithms known to interpolate between SGD and Adam on language modeling (160M parameters, 3.2B SlimPajama tokens, sequence length of 2048, batch size of 256 – Chinchilla optimal). Results report mean and 2-sigma confidence of validation perplexity (100M held-out tokens) over 3 network initialization seeds. Weight decay is always decoupled [21] and set to* 0.1 *[4, 19] except for SGD where we further tune (§C). RMSprop does not use momentum, and Gclip is global norm clipping to* 1 *(before applying momentum), Cclip is coordinate-wise clipping (after applying momentum). Other hyperparameters, for all other methods, are carefully tuned, see e.g. Figure 2 and §B.*

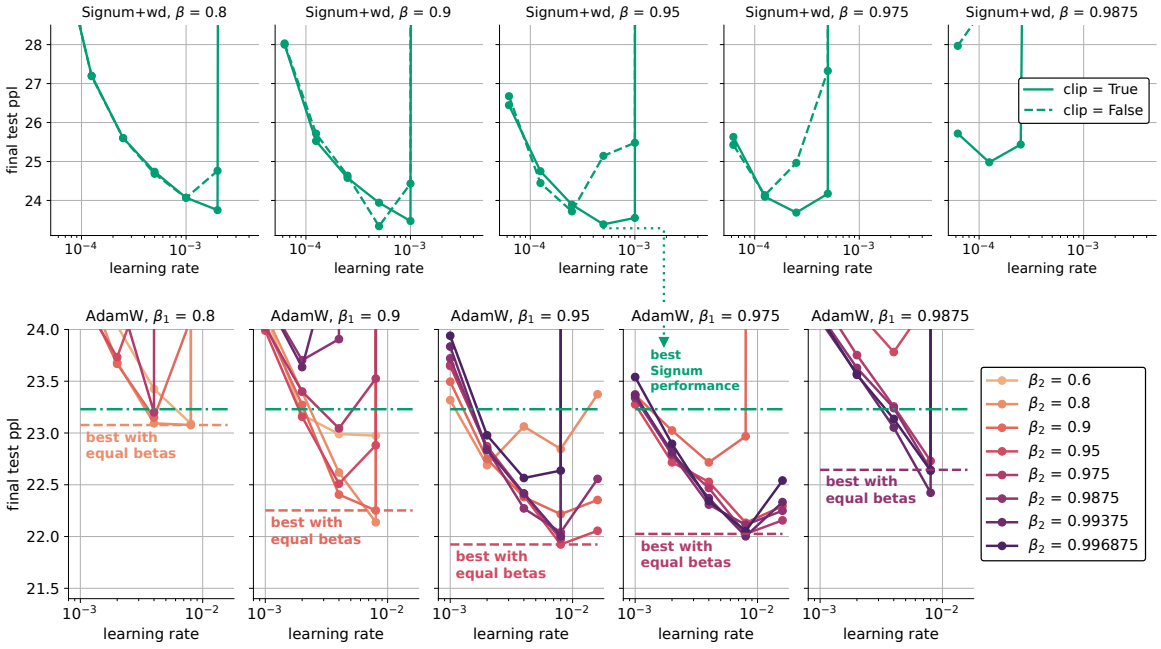

Figure 2: *Training a total of **265 language models** with **160M** parameters with **3.2B** SlimPajama-627B tokens, sequence length of 2048, batch size of 256. Shown is the final test perplexity on 100M held-out tokens. Some underperforming runs are not shown to keep focus on the most interesting range. For a careful description of our tuning grid, see §B. **Takeaway 1:** Validation perplexity of highly tuned (65 hyperparameter configurations) Signum with weight decay 0.1 – top row – is around 23.23 (see Table 1 for multiple seeds at optimal tuning). We ablate on the momentum parameter, learning rate, and presence of global clipping before averaging. The best performance of Signum is reported as a green horizontal line on the second row (200 Adam runs, with weight decay of* 0.1*). Most Adam runs perform better than optimally tuned Signum. **Takeaway 2:** For each β_1, the optimal corresponding β_2 (after learning rate tuning) is similar. The higher β_1, the higher β_2 for optimal performance (optimal βs are correlated).*

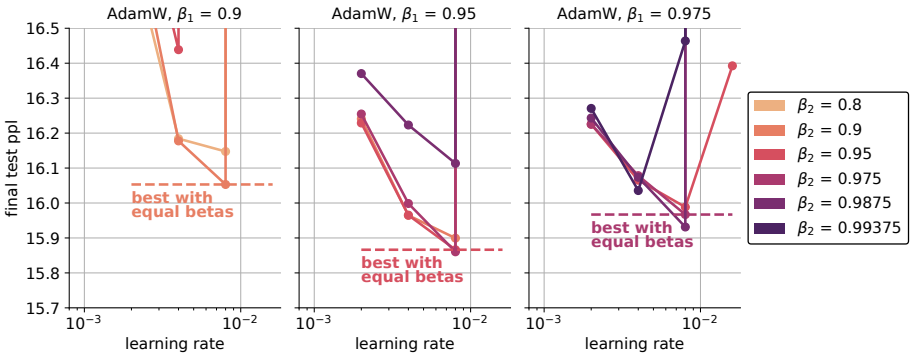

Figure 3: *The final validation performance (100M held-out tokens) for 44 trained LMs with 420M parameters trained on 8.2 B SlimPajama tokens (Chinchilla-optimal).* ***Equal betas yields near-optimal performance****. We use gradient clipping and a batch size of 512 (scaled by 2 compared to Figure 2, as suggested by Zhang et al. [45]). Sequence length is 2048, weight decay is* 0.1.

2. Findings

We present here our two main findings. For a description of our setup see §B.

Takeaway 1. From the results in Figure 2 and Table 1, we observe that optimal tuning of `Signum` with weight decay substantially improves SGD performance [16, 51]. However, the majority of `Adam` runs still achieve superior results, suggesting that a critical component remains—what one might call the "secret sauce"—that continues to give `Adam` an edge over better-understood optimizers in large-scale settings. This gap is not limited to this specific setup. In §C we discuss results on *another dataset (Fineweb), with disabled weight decay, and shorter sequence lengths*. Further, we ablate on other potential confounders (initialization, bias corrections, Adam ϵ value) in §C.5.

Takeaway 2 (a). In Figure 2, we clearly see that $\beta_1 = \beta_2$ yields near-optimal performance in `Adam`, for the five β_1 values we considered. In § C we show similar results at different batch sizes, different sequence lengths, and with disabled weight decay on a different dataset. We also extend this observation to 410M parameters models (Figure 3). This empirical finding serves as a basis for our theory in §3.

Takeaway 2 (b). As a corollary to Takeaway 2, Figure 2 shows that the best performance is not only achieved when $\beta_1 = \beta_2$, but also improves as the two values become closer. Among an additional set of 500 runs on 160M-parameter models (§C.3), we observe a clear correlation: lower loss is associated with smaller differences between β_1 and β_2. This suggests that gradient smoothing (β_1) and preconditioner smoothing (β_2) should not be treated as independent operations—optimal performance often arises when they act in concert.

Standard choice for betas. While in standard deep learning (also Pytorch default) $\beta_2 > \beta_1$ $(0.999, 0.9)$, in language modeling the choice $\beta_1 = 0.9, \beta_2 = 0.95$ is much more common. A lower value for β_2 was shown to help mitigate loss spikes [6, 9], and several recent studies have started to adopt $\beta_1 = \beta_2 = 0.95$ as a default [30, 45, 51]. All our findings confirm this choice for tuning (see e.g. Figure 2), of which we evaluate validity extensively for several values of β_1.

Theoretical relations between betas. We note that a correlation between β parameters was also noted in the theoretical setup of Zhang et al. [48], where it is shown that if β_2 is large enough and

$\beta_1 < \sqrt{\beta_2}$, Adam converges to the neighborhood of critical points. Further, Xie and Li [41] showed that weight decay in AdamW leads to convergence to a constrained minimizer only if $\beta_2 > \beta_1$.

3. New Viewpoints of Adam

We now show that restricting to the case $\beta_1 = \beta_2 = \beta$ yields a useful interpretation of the Adam. Since the Adam update is coordinate-wise, it suffices to analyze a single scalar gradient $g_k \in \mathbb{R}$. Moreover, ablations (Table 2, Table 3) indicate that neither the ϵ-term nor the bias correction significantly affect performance. Thus, for clarity, we set $\epsilon = 0$ and study a simplified update:

$$d_k = \frac{\texttt{EMA}_\beta[g_k]}{\sqrt{\texttt{EMA}_\beta[g_k^2]}}. \tag{1}$$

We next rewrite the update in a form that explicitly highlights the role of variance.

Let $m_k = \texttt{EMA}_\beta[g_k]$. Then the Adam update admits the equivalent representation (see §D):

$$d_k = \frac{m_k}{\sqrt{m_k^2 + \beta\,\texttt{EMA}_\beta[(m_{k-1} - g_k)^2]}}. \tag{2}$$

This shows that the denominator depends on the exponential moving average of the squared deviation between the momentum m_{k-1} and the incoming gradient g_k, with an **interesting multiplier** β. As we demonstrate in the next section, this quantity is an online estimator of the gradient variance.

Comparison with Balles and Hennig [2018]. Setting $\beta_1 = \beta_2$, a choice that was not common[1] at the time of Balles and Hennig [1] allows for precise calculations and claims around variance estimation. In particular, the quantity $\texttt{EMA}_{\beta_2}[g_k^2] - \texttt{EMA}_{\beta_1}[g_k]^2$ allows for a precise variance interpretation only for the case $\beta_1 = \beta_2$: we prove in §D.2 that for any $a, b, \gamma \in \mathbb{R}$ and $\tau \in (0, 1)$, the representation $d_k = \frac{m_k}{\sqrt{m_k^2 + \gamma\,\texttt{EMA}_\tau[(am_{k-1} - bg_k)^2]}}$ holds for Adam **if and only if** $\beta_1 = \beta_2$.

3.1. Adam Estimates Mean and Variance using Variational Inference

We show that Adam admits a natural interpretation as online variational inference, in which

$$m_k := \texttt{EMA}_\beta[g_k] \quad \text{and} \quad \sigma_k := \beta\,\texttt{EMA}_\beta[(m_{k-1} - g_k)^2]$$

correspond to online estimates of the mean and variance of the stochastic gradients. We reintroduce Adam through this lens. Suppose we are given a sequence of stochastic gradients $\{g_1, \ldots, g_k\}$, where each g_k is sampled from an unknown Gaussian distribution whose mean and variance may vary with k. Rather than taking steps directly along these noisy gradients, we aim to estimate their mean and variance online and use these estimates to define a more informed search direction.

At iteration k, let (m_k, σ_k^2) denote our current estimates of the gradient mean and variance, respectively. Upon receiving a new gradient sample $g_{k+1} \sim \mathcal{N}(m, \sigma^2)$ with unknown (m, σ^2), we wish to update our estimates to $(m_{k+1}, \sigma_{k+1}^2)$ so that it becomes more *likely* that g_{k+1} was drawn from $\mathcal{N}(m_{k+1}, \sigma_{k+1}^2)$. Since we also expect the underlying distribution to vary slowly over time, we prefer that $\mathcal{N}(m_{k+1}, \sigma_{k+1}^2)$ remain close to the previous estimate $\mathcal{N}(m_k, \sigma_k^2)$. These two

1. Default parameters were $\beta_1 = 0.9$, $\beta_2 = 0.999$.

goals—fitting the new observation and ensuring smooth updates—can be traded off via a regularized maximum likelihood problem:

$$\min_{m,\sigma^2\geq 0} \; -\log p(g_{k+1} \mid m, \sigma^2) + \tfrac{1}{\lambda}\mathrm{KL}\left(\mathcal{N}(m_k, \sigma_k^2) \,\|\, \mathcal{N}(m, \sigma^2)\right), \tag{3}$$

where $p(g_{k+1} \mid m, \sigma^2)$ is the Gaussian likelihood, $\lambda \geq 0$ is a regularization parameter, and KL denotes the Kullback–Leibler divergence:

$$-\log p(g_{k+1} \mid m, \sigma^2) = \frac{1}{2}\log\sigma^2 + \frac{1}{2\sigma^2}(g_{k+1} - m)^2, \tag{4}$$

$$\mathrm{KL}\left(\mathcal{N}(m_k, \sigma_k^2) \,\|\, \mathcal{N}(m, \sigma^2)\right) = \frac{1}{2}\left[\frac{\sigma_k^2}{\sigma^2} + \frac{(m_k - m)^2}{\sigma^2} - 1 - \log\left(\frac{\sigma_k^2}{\sigma^2}\right)\right]. \tag{5}$$

The following result characterizes the solution of (3), showing that the moving averages used in `Adam` correspond exactly to an instance of online variational inference (see Appendix for a proof):

Let $\beta = \frac{1}{1+\lambda}$. Then the solution to the optimization problem (3) is given by

$$m_{k+1} = \beta m_k + (1-\beta) g_{k+1} = \mathtt{EMA}_\beta(g_{k+1}), \tag{6}$$

$$\sigma_{k+1}^2 = \beta\sigma_k^2 + \beta(1-\beta)(m_k - g_{k+1})^2 = \beta\,\mathtt{EMA}_\beta\left((m_k - g_{k+1})^2\right). \tag{7}$$

As a consequence, the `Adam` update direction in (2) can be rewritten as

$$d_k \;=\; \frac{m_k}{\sqrt{m_k^2 + \beta\mathtt{EMA}_\beta((m_{k-1} - g_k)^2)}} \;=\; \frac{m_k}{\sqrt{m_k^2 + \sigma_k^2}} = \frac{\mathrm{sign}(m_k)}{\sqrt{1 + \sigma_k^2/m_k^2}}. \tag{8}$$

This shows that `Adam` may be interpreted as an *adaptive mollified* variant of `Signum`, where the mollification depends on the local noise-to-signal ratio. This mollified viewpoint aligns well with one of the first papers on understanding `Adam` [1], as discussed in the last section.

Trust regions and norms. Using these insights, we can better formalize the *noise-to-signal* interpretation of `Adam` [1]. Let the quantity m_k/σ_k denote the signal-to-noise ratio (SNR). We show that `Adam` can be viewed as a steepest descent method whose trust region is modulated by the SNR.

To build this connection, consider first the `Signum` update. It corresponds to the steepest descent direction under an ℓ_∞-norm constraint [1], solving $-\mathrm{sign}(m_k) = \mathrm{argmin}_{\theta\in\mathbb{R}} -m_k \cdot \theta$ subject to $|\theta| \leq 1$. That is, `Signum` selects the direction most aligned with $-m_k$ within a unit trust region. In contrast, `Adam` can be interpreted as a steepest descent method with a variable trust region, defined by the (inverse) signal-to-noise ratio:

$$-\frac{\mathrm{sign}(m_k)}{\sqrt{1 + \sigma_k^2/m_k^2}} \;=\; \mathrm{argmin}_{\theta\in\mathbb{R}} -m_k \cdot \theta \quad \text{subject to } |\theta| \leq \frac{1}{\sqrt{1 + \sigma_k^2/m_k^2}}. \tag{9}$$

The effective step size shrinks when the noise dominates the signal, and expands as uncertainty decreases: `Adam` adapts its update magnitude according to a confidence-weighted trust region.

Further insights. To gain deeper insights, we present additional visualizations on the smoothing action of `Adam` in §F. To conclude, we also reproduce most of our finding on the heterogeneous quadratic example by [49], see §E. This example shows both a significant gap between `Adam` and `Signum` and the importance of choosing adaptive mollifiers.

Conclusion. We have presented an extensive numerical study of `Adam`, comparing it against several proposed simplifications. Our main finding is that, on generative language modeling tasks, `Adam` significantly outperforms these simplified variants. Notably, we observe that setting $\beta_1 = \beta_2$ is often optimal. Based on this observation, we recommend `Adam` with $\beta_1 = \beta_2$ as a simplified model, and we provide a new variational inference interpretation for this setting.

Acknowledgements. We thank Niccolo Ajroldi, Sam Liang, and Enea Monzio Compagnoni for their comments. Antonio Orvieto acknowledges the financial support of the Hector Foundation.

References

[1] Lukas Balles and Philipp Hennig. Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients. In *ICML*, 2018.

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

[3] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *ICML*, 2018.

[4] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *ICML*, 2023.

[5] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

[6] Matias D. Cattaneo and Boris Shigida. Tuning adam(w): Default β_2 may be too large, 2025. URL https://mdcattaneo.github.io/papers/Cattaneo-Shigida_2025_TuningAdam.pdf.

[7] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023.

[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] Enea Monzio Compagnoni, Tianlin Liu, Rustem Islamov, Frank Norbert Proske, Antonio Orvieto, and Aurelien Lucchi. Adaptive methods through the lens of SDEs: Theoretical insights on the role of noise. In *ICLR*, 2025.

[10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35, 2022.

[11] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[12] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[13] Andrej Karpathy. Nanogpt, 2022.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems*, 2019.

[16] 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 *ICLR*, 2023.

[17] Frederik Kunstner, Alan Milligan, Robin Yadav, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why adam outperforms gradient descent on language models. *Advances in Neural Information Processing Systems*, 2024.

[18] Bingrui Li, Jianfei Chen, and Jun Zhu. Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36:15136–15171, 2023.

[19] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[20] Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*, 2023.

[21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[22] Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. On the sdes and scaling rules for adaptive gradient algorithms. *Advances in Neural Information Processing Systems*, 2022.

[23] Toan Q Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*, 2019.

[24] Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurelien Lucchi. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. *Advances in Neural Information Processing Systems*, 2022.

[25] Antonio Orvieto, Jonas Kohler, Dario Pavllo, Thomas Hofmann, and Aurélien Lucchi. Vanishing curvature in randomly initialized deep relu networks. In *AISTATS*, pages 7942–7975, 2022.

[26] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

[27] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[29] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[30] Ishaan Shah, Anthony M Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J Shah, et al. Practical efficiency of muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

[31] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[32] Umut Simsekli, Levent Sagun, and Mert Gurbuzbalaban. A tail-index analysis of stochastic gradient noise in deep neural networks. In *ICML*, 2019.

[33] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

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

[35] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 6, 2012.

[36] Akiyoshi Tomihari and Issei Sato. Understanding why adam outperforms sgd: Gradient heterogeneity in transformers. *arXiv preprint arXiv:2502.00213*, 2025.

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] Ben Wang and Aran Komatsuzaki. Gpt-j-6b: A 6 billion parameter autoregressive language model. 2021. *URL https://github. com/kingoflolz/mesh-transformer-jax*, page 8, 2022.

[40] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, 30, 2017.

[41] Shuo Xie and Zhiyuan Li. Implicit bias of adamw: ℓ_∞-norm constrained optimization. In *ICML*, 2024.

[42] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pages 10524–10533. PMLR, 2020.

[43] Wei Yuan and Kai-Xin Gao. Eadam optimizer: How ϵ impact adam. *arXiv preprint arXiv:2011.02150*, 140, 2020.

[44] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

[45] Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham M. Kakade. How does critical batch size scale in pre-training? In *ICLR*, 2025.

[46] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *ICLR*, 2020.

[47] Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank Reddi, Sanjiv Kumar, and Suvrit Sra. Why are adaptive methods good for attention models? *Advances in Neural Information Processing Systems*, 33:15383–15393, 2020.

[48] Yushun Zhang, Congliang Chen, Naichen Shi, Ruoyu Sun, and Zhi-Quan Luo. Adam can converge without any modification on update rules. *Advances in Neural Information Processing Systems*, 2022.

[49] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need adam: A hessian perspective. In *Neural Information Processing Systems*, 2024.

[50] Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik P Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more, 2024.

[51] Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham M Kakade. Deconstructing what makes a good optimizer for autoregressive language models. In *ICLR*, 2025.

[52] Nicolas Zucchet and Antonio Orvieto. Recurrent neural networks: vanishing and exploding gradients are not the end of the story. *Advances in Neural Information Processing Systems*, 2024.

Appendix A. Preliminaries and Related Works

For a signal $(s_k)_{k\in\mathbb{N}}$ and $\beta \in [0, 1)$, we define the β-normalized exponential moving average:

$$\texttt{EMA}_\beta[s_k] = \beta\texttt{EMA}_\beta[s_{k-1}] + (1-\beta)s_k, \qquad \texttt{EMA}_\beta[s_0] := s_0 \text{ (or zero).} \tag{10}$$

The `Adam` optimizer [14] without bias correction [2] takes the following form:

$$w_{k+1} = w_k - \eta_k \left(\sqrt{\texttt{EMA}_{\beta_2}[g_k^2]} + \epsilon\right)^{-1} \texttt{EMA}_{\beta_1}[g_k] \tag{Adam}$$

where all division and multiplications are element-wise, $w_k, g_k \in \mathbb{R}^d$ are model parameters and gradients at iteration k, η_k is the scheduled learning rate, and $\epsilon > 0$ is a small constant. `RMSprop` [35] is an earlier method that sets $\beta_1 = 0$.

One special case, and simplification, of `Adam` is to consider $\beta_1 = \beta_2 = \epsilon = 0$ which gives `SignSGD`:

$$w_{k+1} = w_k - \eta_k \texttt{sign}[g_k]. \tag{SignSGD}$$

A practical variant of `SignSGD`, shown successful in language modeling [16], computes an `EMA` (or momentum) on the gradients before applying the `sign` [3]:

$$w_{k+1} = w_k - \eta_k \texttt{sign}[\texttt{EMA}_\beta[g_k]]. \tag{Signum}$$

In practice, every language modeling pipeline (see e.g. [13]) incorporates some gradient clipping strategy [26], a component known to stabilize training in the autoregressive setting and to make gradients more robust to the stochasticity of language [47]. Global norm clipping (that we abbreviate Gclip), processes gradients fresh out of the backward pass:

$$\texttt{clip}[g_k] = \min\left\{1, \frac{1}{\|g_k\|_2}\right\} g_k.$$

In our experiments, we start from vanilla SGD with momentum: $w_{k+1} = w_k - \eta_k \texttt{EMA}_\beta[g_k]$ and ablate on the positive effect of Gclip before applying momentum. Regarding coordinate clipping (Cclip), a softer version of `sign`, we consider applying it to $\texttt{EMA}_\beta[g_k]$ – in connection with `Signum`.

Research on Adam, a short summary. Despite initial concerns on generalization [40] and convergence [29], after the introduction of decoupled weight decay (i.e., `AdamW` [21]) `Adam` rapidly became the de-facto standard optimizer in deep learning, with works highlighting its landscape adaptation properties [25] and its debated connections to empirical Fisher preconditioning [15].

With the advent of Transformers [38], early works noticed an intriguing gap with `SGD` performance in language modeling [42] (much larger than what can be observed, e.g., in CNNs on image data), that was initially attributed to heavy-tail noise in text data [32, 46] – suggesting `Adam` performance to be correlated with its adaptive coordinate clipping mechanism [46].

As models became larger and more hardware-demanding, interest spiked in the community to reduce the memory footprint of `Adam` [18, 50] and to search for more efficient options [7, 20]. Current trends, draw an intriguing connection between `Adam` and `SignSGD` [2], and in particular

2. We show in Table 3 that the presence of bias correction does not affect our results at the best hyperparameter configuration. However, for all our runs, we use the Pytorch implementation including this factor, for simplicity.

with its momentum variant: `Signum` [3]. This connection was first suggested in early attempts to understand `Adam`'s empirical performance [1], and has recently gained renewed attention in light of transformer architectures and their heterogeneous optimization landscapes [17, 24, 36, 49, 51]. These landscape-based arguments are now more compelling, as recent evidence shows that `Adam` and signed momentum methods outperform SGD even in deterministic settings [16].

Our approach. While we find an abundance of connections between `Adam` and simpler methods such as `Signum` (with less hyperparameters) in the modern literature, the compute requirements for comprehensive research on `Adam` on small to medium scale language models is prohibitive for most academic optimization researchers, especially given the combinatorial exploding hyperparameter grid needed for bulletproof comparisons. Our efforts in §2 are therefore devoted to delivering a comprehensive reference for optimizer performance in many LM settings. We summarize findings in two main takeaways (Figure 2) and provide theoretical investigations (§3) based on our findings.

Appendix B. Experimental details

For pre-training Transformers on Causal Language Modeling, we build upon the nanoGPT [13] implementation, augmenting it with Rotational Positional Embedding [34], RMSNorm [44], and SwiGLU [31]. All our models have a vocabolary size of 50280 and make use of GPT-Neox tokenizer [5]. We adopt an enhanced training recipe, made popular by large language models such as LLaMa [37]. These modifications include: training in bfloat16; employing a linear learning rate warm-up for 10% of the training steps, followed by cosine annealing to $1e-5$. Clipping is used (unless specified) for gradient norms above 1; we have no weight tying between embedding and last linear layer. We always report validation perplexity on a separate subset consisting of 100M tokens. Seeds, when provided, are relative to distinct network initialization.

Computational Resources. All our experiments at a 160M parameter scale are performed on a single NVIDIA A100-SXM4-80GB. At compute optimality (most of our experiments) each run takes approximately 5.83 hours. Our runs at a 410M parameter scale are performed on 8 NVIDIA A100-SXM4-80GB GPUs, each run takes approximately 4.83 hours. For all our runs, we fill up memory and optimize for the gradient accumulation steps.

Code. All our runs use the repository

`https://github.com/Niccolo-Ajroldi/plainLM`

Model settings (160 M). We use the same setting as [4], configuration can be found here: `https://github.com/EleutherAI/pythia/blob/main/models/160M/pythia-160m.yml`

- *Layers:* 12 Transformer [38] layers
- *Attention heads:* 12
- *Hidden size:* 768
- *Attention implementation:* Flashattention [10].
- *MLP type:* SwiGLU [31] with expansion factor 8/3.
- *Backbone:* PreLN transformer [42].

- *Normalization:* RMSnorm [44] for both Attention and MLP.
- *Position embeddings:* Rotary embeddings (RoPE) to 25% of dimensions ([34])
- *Initialization:* the MLP and Attention output weights are initialized with variance $0.02/\sqrt{2\#\text{layers}}$ (scaling also similar to [28]). All other weights (comprising embeddings) are initialized with a standard deviation of 0.02 (Nguyen and Salazar [23], Wang and Komatsuzaki [39], Sec. 2.2). Biases are always initialized at zero.
- *Precision:* Mixed precision FP16 enabled.
- *Dropout:* Disabled for both hidden and attention layers (see also Chowdhery et al. [8]).

Model settings (410 M). We use the same setting as [4], configuration can be found here: https://github.com/EleutherAI/pythia/blob/main/models/410M/pythia-410m-deduped.yml

- *Layers:* 24 Transformer layers
- *Attention heads:* 16
- *Hidden size:* 1024
- Other settings as 160M parameters.

B.1. Experiments on SlimPajama – 160M parameters model

We present 3 experiments sections:

- Section B.1.1 – core setting, ablating on all optimizers.
- Section B.1.2 – ablating on a smaller sequence length.
- Section B.1.3 – ablating at different batch sizes.

All our experiments here use the Cerebras SlimPajama-627B [33] dataset: https://huggingface.co/datasets/cerebras/SlimPajama-627B.

B.1.1. SEQUENCE LENGTH 2048, BATCH SIZE 256, 3.2 B TOKENS (6200 GRADIENT STEPS)

This setup comprises a total of 570 full training runs. We always use warm-up (10%) and cosine anneal until a learning rate of $1e-5$. This setting is Chinchilla-optimal (20 tokens/parameter).

Core experiments: These are the core experimental results for this setting.

- **SGD** with momentum β and **global clipping** to 1 (Gclip, dampening to $1-\beta$) — *84 full runs* (Figure 4, top).

$$\begin{aligned}(\eta, \beta, \lambda) \in & [2.0, 1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125] \\ & \times [0.9875, 0.975, 0.95, 0.9] \\ & \times [0, 1e-3, 1e-4].\end{aligned}$$

- **SGD** with momentum β and (1) **global clipping** of raw gradient to 1 (Gclip) and (2) **coordinate clipping** (Cclip) to 1 after momentum is applied. Dampening is set to $1-\beta$, λ (weight decay) is set to 0
  — *24 full runs* (Figure 4, bottom).

$$(\eta, \beta, \lambda) \in [2.0, 1.0, 0.5, 0.25, 0.125, 0.0625] \times [0.9875, 0.975, 0.95, 0.9]$$

- **SGD** with momentum β (vanilla, dampening to $1-\beta$, **no clipping**). $\lambda = 0$ (weight decay).
  — *28 full runs* (Figure 5)

$$(\eta, \beta) \in [0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125, 0.00390625] \times [0.9875, 0.975, 0.95, 0.9].$$

- **Adam** with global norm clipping to 1 and with $\lambda = 0.1$ (weight decay) and $\epsilon = 1e-8$ (usual setup, see Biderman et al. [4]).
  – *200 full runs* (Figure 2)

$$(\eta, \beta_1, \beta_2) \in [0.016, 0.008, 0.004, 0.002, 0.001] \times [0.9875, 0.975, 0.95, 0.9, 0.8] \times [0.996875, 0.99375, 0.9875, 0.975, 0.95, 0.9, 0.8, 0.6]$$

- **RMSprop** implemented using the AdamW class using $\beta_1 = 0$. We again use $\lambda = 0.1$ (weight decay) and $\epsilon = 1e-8$.
  – *48 full runs* (Figure 6).

$$(\eta, \beta_2) \in [0.004, 0.002, 0.001, 0.0005, 0.00025, 0.000125] \times [0.9875, 0.975, 0.95, 0.9, 0.8, 0.6, 0.4, 0.0]$$

- **Signum** with weight decay $\lambda = 0.1$ as also suggested by [51] (Figure 5, top left panel). We ablate on presence of global norm gradient clipping (to norm 1).
  – *70 full runs* (Figure 2).

$$(\eta, \beta, \text{clip}) \in [0.004, 0.002, 0.001, 0.0005, 0.00025, 0.000125, 0.0000625] \times [0.9875, 0.975, 0.95, 0.9, 0.8] \times [\text{True}, \text{False}]$$

  Note that Signum with and without gradient clipping are two different methods: here, clipped gradients are first averaged and only then the sign is taken. Instead, clipping on the EMA of signed gradients (next method) should have no effect (apart from non-determinism).

- **EMASign** with weight decay $\lambda = 0.1$. We ablate on the presence of global norm gradient clipping (to norm 1) *out of mistake*: the two methods are equal!
  – *70 runs (35 duplicate runs)* (Figure 7)

$$(\eta, \beta, \text{clip}) \in [0.001, 0.0005, 0.00025, 0.000125, 0.0000625, 0.00003125, 0.000015625] \times [0.9875, 0.975, 0.95, 0.9, 0.8] \times [\text{True}, \text{False}]$$

Ablations: These ablations were performed to test side-claims in the paper.

- **Adam** with global norm clipping to 1 and $\lambda = 0.1, \beta_1 = \beta_2 = 0.95, \eta = 0.008$ (best setup from Figure 2). Ablated are
  – *6 full runs* (Table 2).

  $$\epsilon \in [1e-3, 1e-6, 1e-8, 1e-10, 1e-12, 1e-15],$$

  and influence of initializing exponential moving averages to zero (default, ZI) or to the stochastic quantity of interest (gradient initialization, GI). At the same time, we try to remove bias correction. These experiments are presented with 3 random initialization seeds:
  – *9 full runs* (Table 3).

- **Signum** with global norm clipping to 1 and $\lambda = 0.1, \beta = 0.95$ (best setting from Figure 2): we ablate on fixed mollifiers for zero-initialized (ZI) or gradient-initialized (GI) momentum. The mollified we study is $m_k/(\sqrt{m_k} + \epsilon)$:

  $$(\eta, \epsilon) \in [0.001, 0.0005, 0.00025, 0.000125] \times [1e-3, 1e-6, 1e-9]$$

  – *12 full runs* (Table 2). We additionally test the influence of ZI vs. GI with three random seeds at $\epsilon = 0$.
  – *5 full runs* (Table 3).

Other: for the best-performing variants of core experiments, we initialize the model with two other random seeds. This accounts for
– *14 additional full runs* (Table 1).

B.1.2. SEQUENCE LENGTH 512, BATCH SIZE 256, 3.2 B TOKENS (24800 GRADIENT STEPS)

This setup comprises a total of 55 full training runs. We test our core claims (Signum underperforms Adam, $\beta_1 = \beta_2$ works well) at a smaller sequence length. Setting is exactly the same as §B.1.1 for all methods, unless stated otherwise.

- **Adam**, we limit this ablation to $\beta_1 = 0.95$,

  $$(\eta, \beta_2) \in [0.001, 0.002, 0.004, 0.008, 0.016] \times [0.99375, 0.9875, 0.975, 0.95, 0.9, 0.8]$$

  – *25 full runs* (Figure 8).

- **Signum**, we do a full ablation using global norm gradient clipping to 1.

  $$(\eta, \beta) \in [0.0000625, 0.000125, 0.00025, 0.0005, 0.001, 0.002] \times [0.9875, 0.975, 0.95, 0.9, 0.8]$$

  – *30 full runs* (Figure 8).

B.1.3. SEQUENCE LENGTH 2048, VARIABLE BATCH SIZE, 2.5 B TOKENS

We use here a slightly reduced token budget (2.5B, 20 tokens for every non-embedding parameter) and run the same Adam tuning experiment presented in Figure 2 for batch size 256. We actually run this experiment again at a batch size of 256, and test batch sizes of 128 and 512 reducing or doubling the number of steps accordingly (same token budget). The sequence length is still 2048, and the dataset SlimPajama. Due to the reduced number of tokens, each run takes approximately 4.7 hours on our hardware. We implement variation of batch size using gradient accumulation $(4, 8, 16)$ at a micro-batch size of 32 sequences. This setup comprises a total of 500 full training runs.

Adam with $\lambda = 0.1$ (weight decay) and $\epsilon = 1e-8$ (usual setup, see Biderman et al. [4]), we clip gradients to global norm 1.

- For batch size 256:

$$\begin{aligned}(\eta, \beta_1, \beta_2) \in &[0.016, 0.008, 0.004, 0.002, 0.001] \\ &\times [0.9875, 0.975, 0.95, 0.9, 0.8] \\ &\times [0.996875, 0.99375, 0.9875, 0.975, 0.95, 0.9, 0.8, 0.6]\end{aligned}$$

- For batch size 128 and 512:

$$\begin{aligned}(\eta, \beta_1, \beta_2) \in &[0.0005, 0.001, 0.0014, 0.002, 0.0028, 0.004, 0.0056, 0.008, 0.0112, 0.016] \\ &\times [0.975, 0.95, 0.9] \\ &\times 1 - [4, 2, 1, 0.5, 0.25] \cdot (1 - \beta_1) \qquad \text{(i.e. 3 higher and 2 lower values in grid)}\end{aligned}$$

  Note that here we overturned the learning rate, the reason for this is the square root scaling law in Compagnoni et al. [9], Malladi et al. [22]: if batch size scales by 2, learning rate should scale as $\sqrt{2}$. We see in §C.3 that this indeed seems to hold true, yet noise prevents us from making precise verification claims.

– *500 full runs* (§C.3).

B.2. Experiments on SlimPajama – 410M parameters model, 8.2 B tokens

All our experiments here use the Cerebras SlimPajama-627B [33] dataset: https://huggingface.co/datasets/cerebras/SlimPajama-627B. We focus on evaluating whether $\beta_1 = \beta_2$ yields good performance in this settings. We scale up the batch size by a factor 2 compared to Section B.1, as suggested by [45]. We perform our experiments at compute optimality (8.2B tokens, 20 tokens per parameter).

Adam with $\lambda = 0.1$ (weight decay) and $\epsilon = 1e-8$ (usual setup, see Biderman et al. [4]), we clip gradients to global norm 1:

- $\beta_1 = 0.9$

$$(\eta, \beta_2) \in [0.016, 0.008, 0.004, 0.002] \times [0.95, 0.9, 0.8]$$

- $\beta_1 = 0.95$

$$(\eta, \beta_2) \in [0.016, 0.008, 0.004, 0.002] \times [0.9875, 0.975, 0.95, 0.9]$$

- $\beta_1 = 0.975$

$$(\eta, \beta_2) \in [0.016, 0.008, 0.004, 0.002] \times [0.99375, 0.9875, 0.975, 0.95]$$

– *44 full runs* (Figure 3).

B.3. Experiments on Fineweb – 160M parameters model, 3.2B tokens – no weight decay

While testing our claims on a different dataset, we also crucially *remove weight decay* here. Our setting is otherwise identical to that of §B.1.1: Sequence length is 2048, batch size is 256, model has 160 parameters and we train on 3.2B tokens from Fineweb [27] https://huggingface.co/datasets/HuggingFaceFW/fineweb.

- **Adam** with $\lambda = 0$ (no weight decay!) and $\epsilon = 1e-8$ (usual setup, see Biderman et al. [4]). We clip gradients to global norm 1.

$$\begin{aligned}(\eta, \beta_1, \beta_2) \in &[0.032, 0.016, 0.008, 0.004, 0.002, 0.001] \\ &\times [0.975, 0.95, 0.9] \\ &\times [0.9875, 0.975, 0.95, 0.9, 0.8]\end{aligned}$$

  – *90 full runs* (Figure 12)

- **Signum** with $\lambda = 0$ (no weight decay) as also suggested by [51] (Figure 5, top left panel). We clip gradients to global norm 1.

$$\begin{aligned}(\eta, \beta) \in &[0.004, 0.002, 0.001, 0.0005, 0.0000625, 0.00025, 0.000125] \\ &\times [0.975, 0.95, 0.9]\end{aligned}$$

  – *24 full runs* (Figure 12).

Appendix C. Complementary Experimental Results

The results in this section complement the discussion in §2. We organize them in 5 subsections, and report all technical details in §B.

- §C.1 outlines all hyperparameter tuning curves for the setting in Table 1 for `SGD` (with/without clipping and with/without weight decay) – Figure 4 and 5, `RMSprop` without momentum – Figure 6, and momentum on top of `SignSGD` – Figure 7.
- §C.3 validates that $\beta_1 = \beta_2$ is a strong-performing option for `Adam` across different batchsizes.
- §C.2 validates that $\beta_1 = \beta_2$ is a strong-performing option for `Adam` at a shorter sequence length. Here, we also show that `Signum` performance is still suboptimal (cf. Figure 2).
- §C.4 reproduces the `Signum-Adam` gap on Fineweb [27]. Compared to Figure 2 and the settings above, here we compare at zero weight decay to eliminate this additional confounder.
- §C.5 confirms on the validity of our findings when ablating on nuances of `Signum` and `Adam` such as initialization and bias correction.

C.1. Tuning for Table 1

Setup Summary. 160 M parameters LM on SlimPajama, trained for 3.2 B tokens at a batchsize of 256 $\times$ 2048 sequence length.

Comment. Our objective here is to tune to best, despite the combinatorially exploding number of options, our methods in Table 1. Details regarding our hyperparameters grid and model configurations are reported in §B. We remind that tuning for `Signum` and `Adam` is presented directly in the main paper as Figure 2. **All figures below show optimal tuning jointly in learning rate and momentum space**. While tuning for `RMSprop` and momentum on `SignSGD` is straightforward, `SGD` requires more attention: we found that removing weight decay was always beneficial when global norm clipping the raw gradient, hence we adopt this option also for the non-clipped variant, and for the variant that includes an additional coordinate clipping step after applying momentum. We believe this is due to the decoupled nature of weight decay, combined with the high learning rates required for good performance in `SGD`.

Finalizing Table 1. After careful tuning, we select for each method the best configuration (given by figures below) and run two additional seeds to report final results with confidence bars.

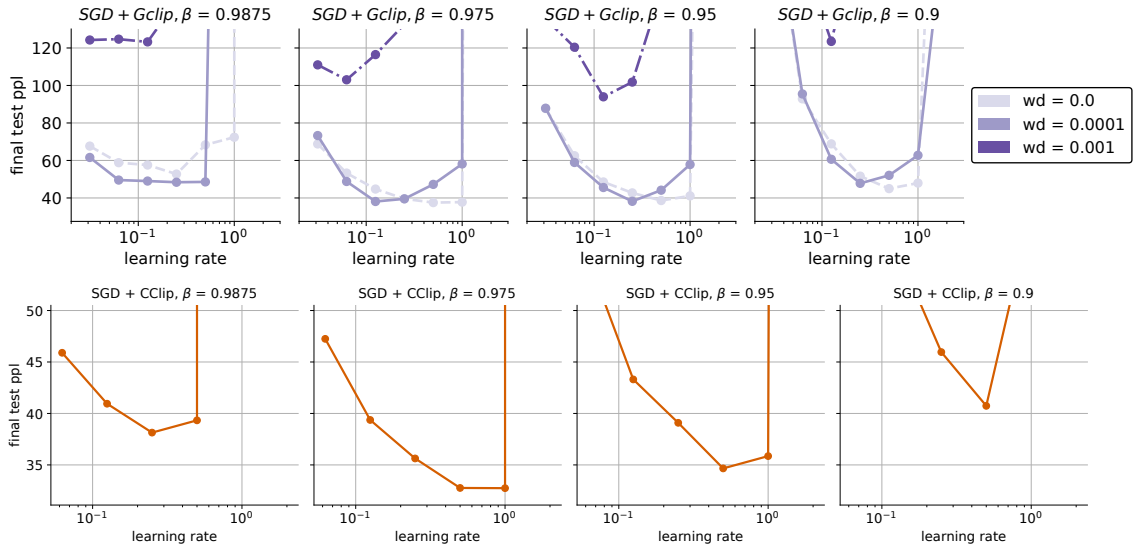

Figure 4: ***(top) SGD with global norm clipping***. *We found it beneficial to remove weight decay: the best setting achieves 37.53 ppl, while a slightly larger wd leads to 38.11. a weights decay of 0.001 is too large and yields 93.7 best validation perplexity.* ***(bottom) SGD with global norm clipping on raw gradients, followed by coordinate clipping on momentum.*** *We remove weight decay as suggested by the top plot. We observe an improvement of 5 perplexity points.*

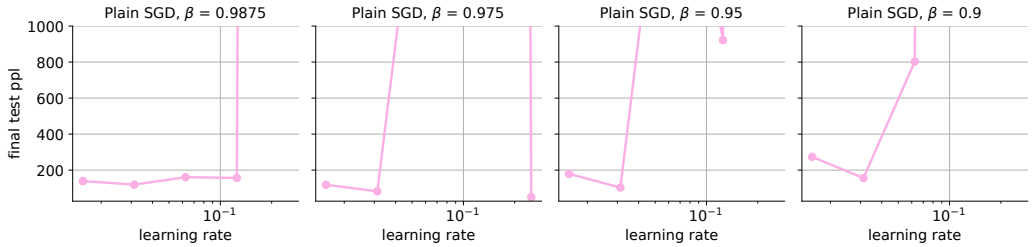

Figure 5: ***SGD without coordinate-wise clipping*** *at zero weight decay (as suggested by Figure 4).*

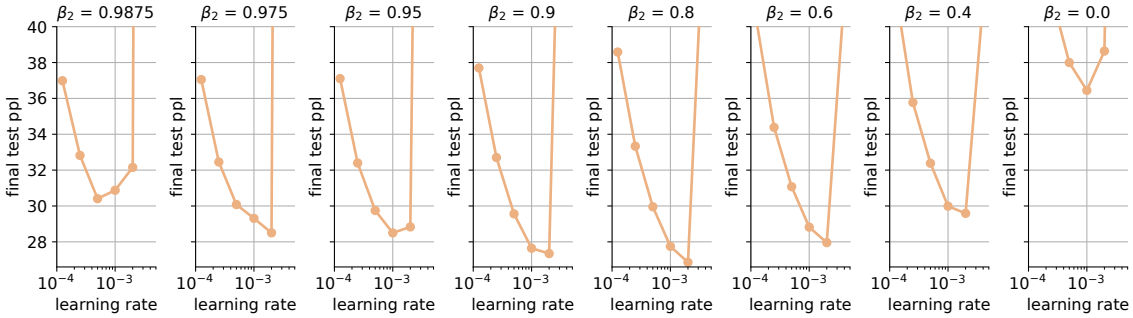

Figure 6: ***RMSprop with decoupled weight decay*** *0.1. Implemented with Pytorch AdamW setting* $\beta_1 = 0$.

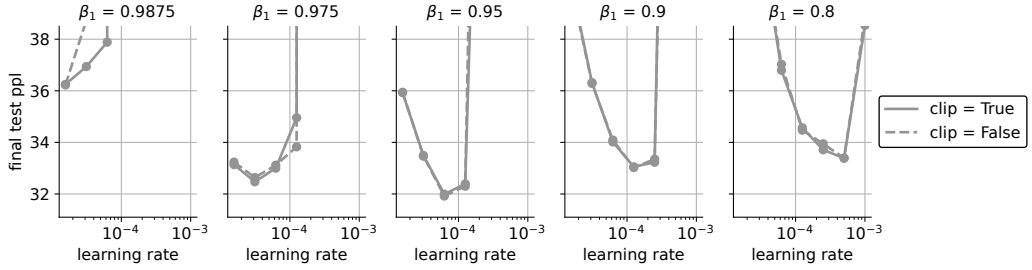

Figure 7: ***Momentum on SignSGD*** *with decoupled weight decay. We implement this just for completeness to show that it is performing worse than* `Signum`*. Clipping has mathematically no effect (we did not notice at first, so we show the result anyways).*

C.2. Effect of Shorter Sequence Length in Figure 2

We run part of the experiments in Figure 2 at a lower sequence length (512), for a batch size of 256 sequences (as Figure 2). The model here still sees 3.2B tokens (compute optimal), but number of effective optimizer steps is 4 times bigger compared to the 2048 sequence length setting. While we still observe a sizeable gap between `Signum` and `Adam`, we note that this is smaller here, as noted also by Zhao et al. [51] in a similar setting.

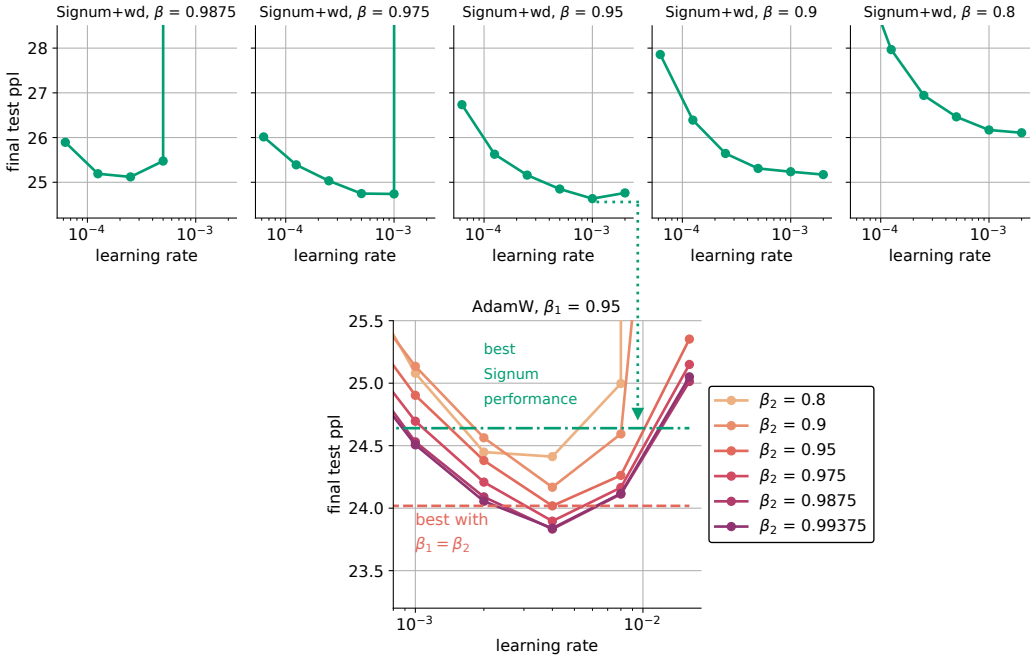

Figure 8: ***AdamW vs Signum****, same setting as Figure 2, but at a smaller sequence length (512).*

C.3. Batch size ablation for Figure 2

We run part of the experiments in Figure 2 at a lower and higher batch size. All other details remain the same and are summarized in §B – except for the number of steps performed: due to limitations in our resources, we chose here to train models for 2.5B tokens – i.e. a slightly undertrained setting (optimal would be 3.2B). In line with [9, 22] we consider half-steps when tuning. All experiments use a weight decay of 0.1.

Despite some imperfections, we notice that $\beta_1 = \beta_2$ is a strong choice even at different batch sizes, our Takeaway 2.

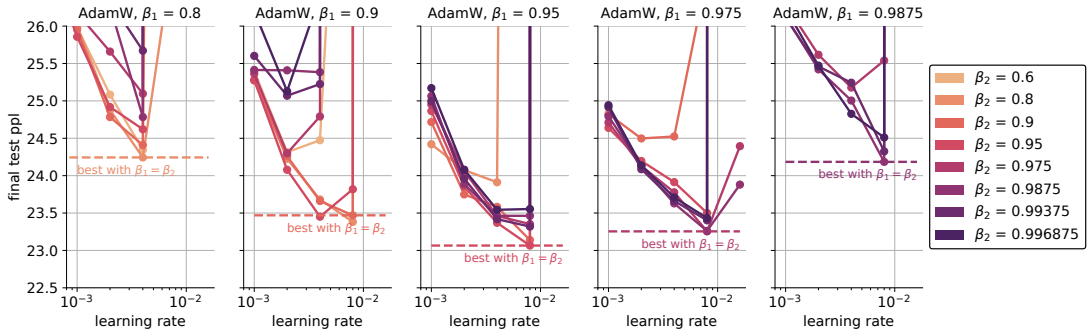

Figure 9: ***Adam, batch size 256*** *trained for 2.5B tokens. Other settings are same setting as Figure 2.*

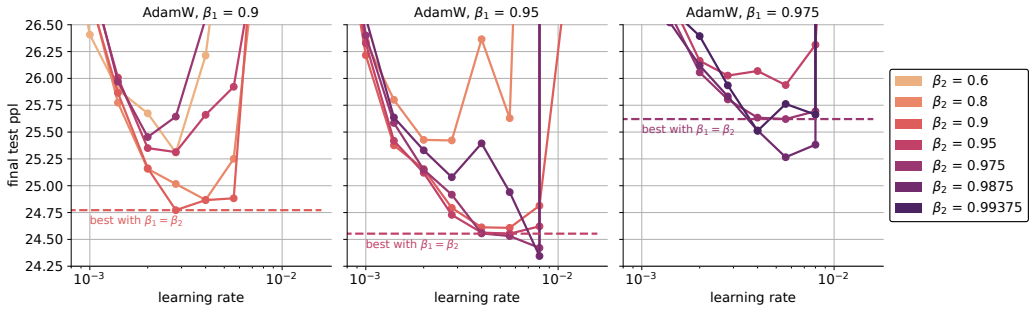

Figure 10: ***Adam, batch size 512*** *trained for 2.5B tokens. Other settings are same setting as Figure 2.*

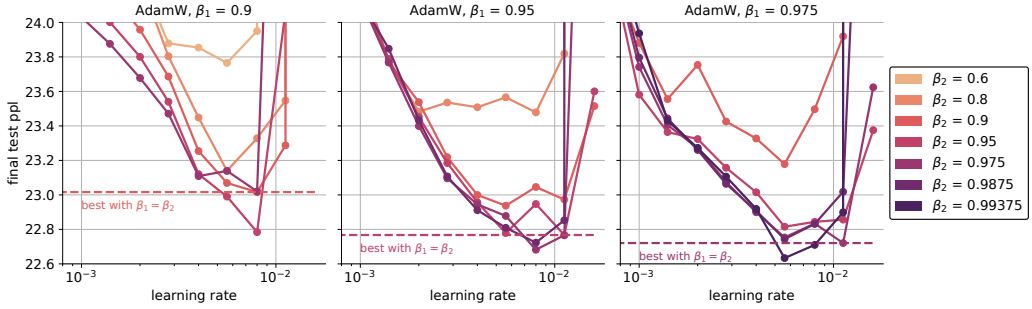

Figure 11: ***Adam, batch size 128*** *trained for 2.5B tokens. Other settings are same setting as Figure 2.*

C.4. Figure 2 on Fineweb (no weight decay)

Finally, we evaluate our findings – both strong performance of equal βs in `Adam` and substantial gap with `Signum` on a different dataset (Fineweb [27]). All other experiments in this paper are performed on SlimPajama. To add an additional axis of variation compared to previously presented settings, we here remove weight decay from all methods.

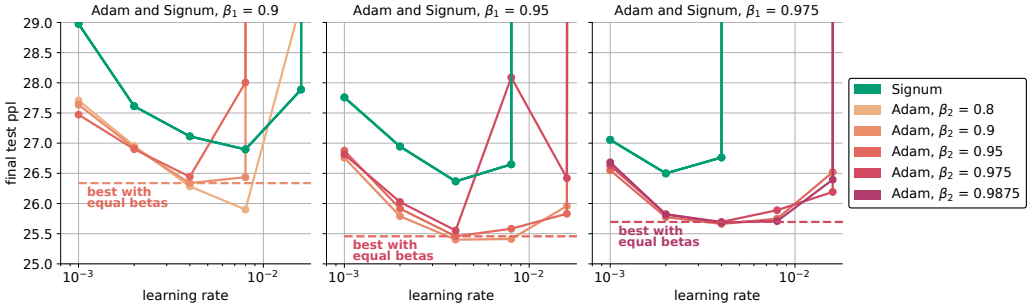

Figure 12: ***`Adam` and `Signum` (no weight decay) on Fineweb***. *Other settings are same as Figure 2.* For visualization purposes, here **we rescaled the visualized learning rate** of `Signum` by a factor ~ 10.

C.5. Effect of Bias Correction and Zero Initialization on `Adam`

When comparing `Signum` with `Adam`, here are a few confounders that might affect results:

The value of epsilon in `Adam` was shown to be important for numerical stability, and might affect performance [43]. We show in Table 2 that one can choose an extremely small ϵ value in our setting. We cross-check the impact of including an ϵ factor in `Signum`: we found that little can be gained from this strategy (Figure 13). In short, we found that ϵ is not a crucial parameter in our setup. This is also liked to our findings on adaptive mollifiers, cf. §3.

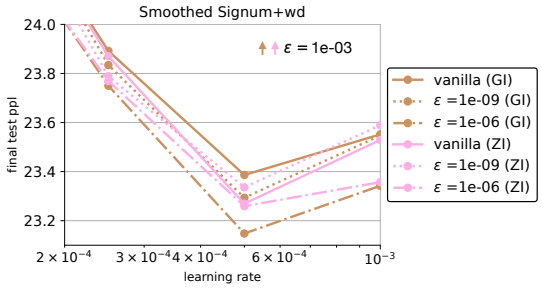

Figure 13: *Adding an ϵ mollifier to `Signum`, i.e., using $m_k/(\sqrt{m_k^2}+\epsilon)$ offered little to no improvement. We also test both zero initialization (ZI) and gradient initialization (GI) for m, and find similar results with no significant improvement.*

Initialization of moving average parameters. In Figure 13 we also ablate on initialization of the moving average in `Signum` and found no substantial differences. We perform this same ablation for `Adam` and report comprehensive results with seeds in Figure 3.

Bias correction. While bias correction in `Adam` is helpful in early training, final validation performance is almost unchanged, see the full training curve and results with seeds in Figure 14.

Table 2: *Effect of ϵ in `AdamW`– other parameters optimally tuned for $\epsilon = 10^{-8}$ (setting: Figure 2). All values between $\epsilon \in [10^{-6}, 10^{-15}]$ result in a similar validation perplexity.*

	$\epsilon = 1e-3$	$\epsilon = 1e-6$	$\epsilon = 1e-8$	$\epsilon = 1e-10$	$\epsilon = 1e-12$	$\epsilon = 1e-15$
Val ppl	23.34± 0.31	21.56± 0.19	21.86± 0.21	21.87± 0.04	21.89± 0.2	21.91± 0.18

Table 3: *ZI denotes Zero init of EMA parameters, GI denotes init of EMA parameters to the measurement at first iteration, BC denotes Bias Correction. **Not** doing ZI means we initialize m and v at g_0 and g_0^2 respectively. Default for `Adam` is ZI and BC. Default for Signum+WD is less clear. We found that initialization does not affect much performance in Signum, yet it does in `Adam`. Performing bias correction is not as important as initialization in `Adam`.*

	`Adam` (+ZI+BC)	`Adam` (+ZI-BC)	`Adam` (+GI-BC)	Signum (+GI)	Signum (+ZI)
Val ppl	21.86± 0.21	21.89± 0.16	22.58± 0.35	23.23± 0.16	23.30± 0.25

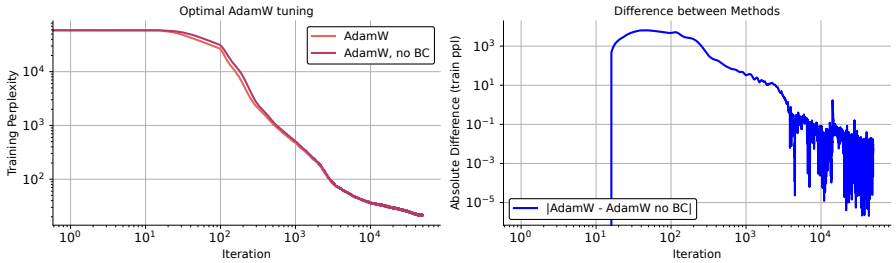

Figure 14: *Effect of eliminating bias correction in `Adam`. The difference between variants vanishes as iterations progress. Plotted is the average dynamics over 3 random seeds.*

Appendix D. Missing proofs and derivations

D.1. Proof of Equation 2

For this proof we will use the abbreviation

$$v_k := \texttt{EMA}_\beta[g_k^2].$$

With this abbreviation the `Adam` update can be written as

$$d_k = \frac{m_k}{\sqrt{v_k}} = \frac{m_k}{\sqrt{m_k^2 + v_k - m_k^2}}.$$

Next we will show that $v_k - m_k^2 = \beta\texttt{EMA}_\beta[(m_{k-1} - g_k)^2]$. Indeed by expanding the update of v_{k+1} and m_{k+1} we have that

$$\begin{aligned}
v_{k+1} - m_{k+1}^2 &= \beta v_k + (1-\beta)g_{k+1}^2 - (\beta m_k + (1-\beta)g_{k+1})^2 \\
&= \beta v_k + (1-\beta)g_{k+1}^2 - (\beta^2 m_k^2 + (1-\beta)^2 g_{k+1}^2 + 2\beta(1-\beta)g_{k+1}m_k) \\
&= \beta v_k - \beta^2 m_k^2 + (1-\beta)\beta g_{k+1}^2 - 2\beta(1-\beta)g_{k+1}m_k \\
&= \beta v_k - \beta m_k^2 + \beta m_k^2 - \beta^2 m_k^2 + (1-\beta)\beta g_{k+1}^2 - 2\beta(1-\beta)g_{k+1}m_k \\
&= \beta(v_k - m_k^2) + \beta(1-\beta)m_k^2 + \beta(1-\beta)g_{k+1}^2 - 2\beta(1-\beta)g_{k+1}m_k \\
&= \beta(v_k - m_k^2) + \beta(1-\beta)(m_k - g_{k+1})^2.
\end{aligned}$$

By setting $\delta_k = v_k - m_k^2$ we have that

$$\delta_{k+1} = \beta\delta_k + \beta(1-\beta)(m_k - g_{k+1})^2 = \beta\texttt{EMA}_\beta[(m_{k-1} - g_k)^2]$$

where we used the definition of the EMA recurrence in (10).

D.2. Necessity of equal betas for variance interpretation

`Adam` with hyperparameters $\beta_1, \beta_2 \in (0,1)$ has update of form

$$\frac{m_k}{\sqrt{m_k^2 + \gamma\texttt{EMA}_\tau[(am_{k-1} - bg_k)^2]}},$$

for some $a, b, \gamma \in \mathbb{R}$ and $\tau \in (0,1)$ if an only if $\beta_1 = \beta_2$.

Let us expand the expression.

$$\begin{aligned}
v_{k+1} - m_{k+1}^2 &= \beta_2 v_k + (1-\beta_2)g_{k+1}^2 - (\beta_1 m_k + (1-\beta_1)g_{k+1})^2 \\
&= \beta_2 v_k + (1-\beta_2)g_{k+1}^2 - [\beta_1^2 m_k^2 + (1-\beta_1)^2 g_{k+1}^2 + 2\beta_1(1-\beta_1)m_k g_{k+1}] \\
&= \beta_2 v_k - \beta_1^2 m_k^2 + [(1-\beta_2) - (1-\beta_1)^2]g_{k+1}^2 - 2\beta_1(1-\beta_1)m_k g_{k+1}
\end{aligned}$$

The case of equal betas. Notice that if $\beta_1 = \beta_2 = \beta$, then

$$(1-\beta) - (1-\beta)^2 = 1 - \beta - (1 + \beta^2 - 2\beta) = 1 - \beta - 1 - \beta^2 + 2\beta = \beta(1-\beta),$$

and so the expression gets simplified:

$$v_{k+1} - m_{k+1}^2 = \beta v_k - \beta^2 m_k^2 + \beta(1-\beta)[g_{k+1}^2 - 2m_k g_{k+1}]$$

Now add and subtract βm_k^2, to get

$$v_{k+1} - m_{k+1}^2 = \beta(v_k - m_{k+1}) + \beta(1-\beta)[m_k^2 + g_{k+1}^2 - 2m_k g_{k+1}].$$

where the last term is the perfect square $(m_k - g_{k+1})^2$.

The general setting. One might hope for the "stars aligning" into a perfect square also in the general setting. For this to happen, we need to require that the term

$$[(1-\beta_2) - (1-\beta_1)^2]g_{k+1}^2 - 2\beta_1(1-\beta_1)m_k g_{k+1}$$

allows for such a simplification to happen. That is, assume to start from

$$(am_k - bg_{k+1})^2 = a^2 m_k - 2abm_k g_{k+1} + b^2 g_{k+1}^2.$$

we need

$$b^2 = (1-\beta_2) - (1-\beta_1)^2, \quad ab = \beta_1(1-\beta_1).$$

so

$$a = \frac{\beta_1(1-\beta_1)}{\sqrt{(1-\beta_2) - (1-\beta_1)^2}}.$$

Therefore:

$$\left(\frac{\beta_1(1-\beta_1)}{\sqrt{(1-\beta_2) - (1-\beta_1)^2}} m_k - \sqrt{(1-\beta_2) - (1-\beta_1)^2}\, g_{k+1}\right)^2$$
$$= \frac{\beta_1^2(1-\beta_1)^2}{(1-\beta_2) - (1-\beta_1)^2} m_k^2 + [(1-\beta_2) - (1-\beta_1)^2]g_{k+1}^2 - 2\beta_1(1-\beta_1)m_k g_{k+1}$$

Therefore, in the general setting, we can write

$$v_{k+1} - m_{k+1}^2 = \beta_2 v_k - \left(\beta_1^2 + \frac{\beta_1^2(1-\beta_1)^2}{(1-\beta_2) - (1-\beta_1)^2}\right) m_k^2 +$$
$$+ \left(\frac{\beta_1(1-\beta_1)}{\sqrt{(1-\beta_2) - (1-\beta_1)^2}} m_k - \sqrt{(1-\beta_2) - (1-\beta_1)^2}\, g_{k+1}\right)^2$$

Massaging a bit, we get

$$v_{k+1} - m_{k+1}^2 = \beta_2 v_k - \frac{\beta_1^2(1-\beta_2)}{(1-\beta_2) - (1-\beta_1)^2} m_k^2 +$$
$$+ \left(\frac{\beta_1(1-\beta_1)}{\sqrt{(1-\beta_2) - (1-\beta_1)^2}} m_k - \sqrt{(1-\beta_2) - (1-\beta_1)^2}\, g_{k+1}\right)^2$$

which implies

$$v_{k+1} - m_{k+1}^2 = \beta_2 \left(v_k - \frac{\beta_1^2(1-\beta_2)}{\beta_2(1-\beta_2) - \beta_2(1-\beta_1)^2} m_k^2 \right) + \\ + \left(\frac{\beta_1(1-\beta_1)}{\sqrt{(1-\beta_2) - (1-\beta_1)^2}} m_k - \sqrt{(1-\beta_2) - (1-\beta_1)^2} g_{k+1} \right)^2 .$$

Therefore, the formula holds true if and only if

$$\frac{\beta_1^2(1-\beta_2)}{\beta_2(1-\beta_2) - \beta_2(1-\beta_1)^2} = 1.$$

That is, if and only if

$$\beta_1^2(1-\beta_2) = \beta_2(1-\beta_2) - \beta_2(1-\beta_1)^2.$$

The condition simplifies, as it reads:

$$\beta_1^2 - \beta_1^2\beta_2 = \beta_2 - \beta_2^2 - \beta_2 - \beta_2\beta_1^2 + 2\beta_1\beta_2.$$

which simplified is

$$\beta_1^2 + \beta_2^2 - 2\beta_1\beta_2 = 0.$$

i.e.

$$(\beta_1 - \beta_2)^2 = 0 \quad \iff \quad \beta_1 = \beta_2.$$

D.3. Proof of the variational perspective

Recall that

$$-\log p(g_{k+1} \mid m, \sigma^2) = \frac{1}{2}\log\sigma^2 + \frac{1}{2\sigma^2}(g_{k+1} - m)^2,$$
$$\mathrm{KL}\left(\mathcal{N}(m_k, \sigma_k^2) \,\|\, \mathcal{N}(m, \sigma^2)\right) = \frac{1}{2}\left[\frac{\sigma_k^2}{\sigma^2} + \frac{(m_k - m)^2}{\sigma^2} - 1 - \log\left(\frac{\sigma_k^2}{\sigma^2}\right)\right].$$

Therefore

$$F(m, \sigma^2) = -\log p(g_{k+1} \mid m, \sigma^2) + \frac{1}{\lambda}\mathrm{KL}\left(\mathcal{N}(m_k, \sigma_k^2) \,\|\, \mathcal{N}(m, \sigma^2)\right) \\ = \frac{1}{2}\log\sigma^2 + \frac{1}{2\sigma^2}(g_{k+1} - m)^2 + \frac{1}{2\lambda}\left[\frac{\sigma_k^2}{\sigma^2} + \frac{(m_k - m)^2}{\sigma^2} - 1 - \log\left(\frac{\sigma_k^2}{\sigma^2}\right)\right]$$

Since we are not optimizing for σ_k^2, we can replace $-\log\left(\frac{\sigma_k^2}{\sigma^2}\right) = \log(\sigma^2)$ and drop constants, gives the following objective function

$$\min_{m,\sigma^2\geq 0} F(m, \sigma^2) = \frac{1}{2}\frac{1+\lambda}{\lambda}\log(\sigma^2) + \frac{1}{2\sigma^2}\left[(g - m)^2 + \frac{1}{\lambda}\left(\sigma_k^2 + (m_k - m)^2\right)\right] + \text{const.}$$

Stationarity in m: Differentiating in m and setting to zero gives

$$\frac{\partial F}{\partial m} = -\frac{1}{\sigma^2}(g-m) - \frac{1}{\lambda\sigma^2}(m_k - m) = 0.$$

Multiplying by $\lambda\sigma^2$, we get:

$$-\lambda(g-m) - (m_k - m) = 0 \quad \Rightarrow \quad m = \frac{\lambda g + m_k}{1+\lambda}. \tag{11}$$

Stationarity in σ^2: Differentiating in σ^2 and setting to zero gives

$$\frac{\partial F}{\partial \sigma^2} = \frac{1}{2}\frac{1+\lambda}{\lambda}\cdot\frac{1}{\sigma^2} - \frac{1}{2\sigma^4}\left[(g-m)^2 + \frac{1}{\lambda}\left(\sigma_k^2 + (m_k - m)^2\right)\right] = 0.$$

Multiplying both sides by $2\sigma^4$, and re-arranging gives:

$$\frac{1+\lambda}{\lambda}\sigma^2 = (g-m)^2 + \frac{1}{\lambda}\left(\sigma_k^2 + (m_k - m)^2\right).$$

Multiplying through by $\frac{\lambda}{1+\lambda}$ gives

$$\sigma^2 = \frac{\lambda(g-m)^2 + \left[\sigma_k^2 + (m_k - m)^2\right]}{1+\lambda}. \tag{12}$$

Now using $m = \frac{\lambda g + m_k}{1+\lambda}$ from (11) we have that

$$g - m = \frac{g - m_k}{1+\lambda}, \quad m_k - m = \frac{\lambda(m_k - g)}{1+\lambda}.$$

Therefore:

$$(g-m)^2 = \frac{(g-m_k)^2}{(1+\lambda)^2}, \quad (m_k - m)^2 = \frac{\lambda^2(g-m_k)^2}{(1+\lambda)^2}.$$

Using the above in the expression for σ^2 in (12), we get:

$$\sigma^2 = \frac{\lambda(g-m_k)^2}{(1+\lambda)^2} + \frac{\sigma_k^2}{1+\lambda}.$$

This, together with (11) gives the final solution

$$\boxed{m_{k+1} = \frac{m_k + \lambda g}{1+\lambda}} \quad \text{and} \quad \boxed{\sigma_{k+1}^2 = \frac{\sigma_k^2}{1+\lambda} + \frac{\lambda(g-m_k)^2}{(1+\lambda)^2}}.$$

If we use the standard momentum parameterization, which corresponds to $\beta = \frac{1}{1+\lambda}$ we arrive at the stated results (6) and (7) of the theorem.

Appendix E. Why an adaptive trust region? Insights from heterogeneous quadratics

While our theoretical analysis in §3 offers a new perspective on `Adam`, it is not tied to any specific architecture. To enhance intuition and provide a controlled setting for future work, we validate our findings on a simplified model of transformer loss landscapes introduced by Zhang et al. [49], building on signal propagation theory [24].

As noted in Kunstner et al. [17], Zhang et al. [49], Zhao et al. [51], the landscape of autoregressive language models is highly heterogeneous: Hessian blocks associated with semantically distinct parameter groups (e.g., normalization layers, embeddings, or softmax-related parameters) exhibit markedly different eigenspectra and thus demand different learning rates. In contrast to homogeneous models (e.g., CNNs), this heterogeneity is where `Adam` significantly outperforms `SGD` [cf. 52].

Figure 15 illustrates this point. On a toy heterogeneous quadratic landscape, tuned `Adam` with equal β values substantially outperforms tuned `SGD` with momentum, echoing results from Zhang et al. [49]. We also observe that `Signum` closes much of the gap but still falls short of `Adam`. This is consistent with our findings in Table 1 for language models.

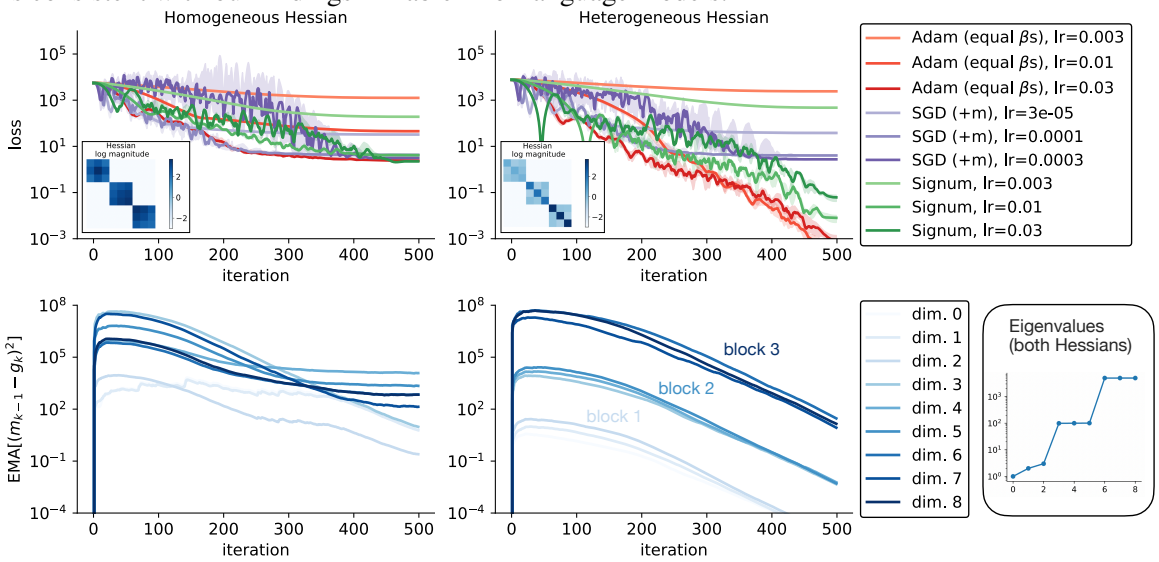

Figure 15: ***Top row:*** *Training performance (median and 25%/75% quantiles over 10 seeds) of* `SGD`*,* `Signum`*, and* `Adam` *on two 9-dimensional convex quadratic problems inspired by Zhang et al. [49]. All optimizers use moving average parameters set to* 0.95*, with a 10% warmup followed by cosine decay to zero. Both problems share the same Hessian eigenspectrum and have a* 3×3 *block structure. The landscape on the* left *is* homogeneous*, with each block containing both large and small eigenvalues. The landscape on the* right *is* heterogeneous*, with each block having eigenvalues of different magnitudes. In this setting,* `Adam` *clearly outperforms* `SGD`*, with* `Signum` *closing part of the gap.* ***Bottom row:*** *Dynamics of the variance term Eq.* (2)*. The value of this term varies both across iterations and across blocks, adapting to the local curvature structure. This adaptive behavior improves performance over* `Signum` *in the heterogeneous setting.*

Setup. Our setup here is inspired directly from the results and discussions in Zhang et al. [49]. Specifically, we consider the loss

$$L(w) = \frac{1}{2} w^\top H w$$

where we construct the Homogeneous and Heterogeneous Hessians using the following procedure:

- We fix the eigenvalues, equal in both cases, to

$$\text{eig}(H_{\text{hom}}) = \text{eig}(H_{\text{het}}) = \{1, 2, 3, 99, 100, 101, 4998, 4999, 5000\}.$$

- We choose both Hessians to be block-diagonal, with blocks of size 3×3. The homogeneous Hessian has eigenvalues of different magnitude in each block, while the Heterogeneous keeps similar magnitudes in each block.

```
H_details_het = [[1,2,3],[99,100,101],[4998,4999,5000]]
H_details_hom = [[1,99,4998],[2,100,4999],[3,101,5000]]
```

- For each block, we apply a random rotation to the diagonal matrix of eigenvalues, specific to each block. Each rotation is sampled from the Haar measure by decomposition of a random 3×3 positive semidefinite matrix $AA^\top$, where $A \in \mathbb{R}^{3\times3}$ has i.i.d. Gaussian entries.

The result is shown in Figure 15.

Next, to introduce stochasticity in this setting, we simply take the square root of the Hessian to define a 9×9 design matrix X

$$H = X^\top X, \qquad X = H^{\frac{1}{2}}$$

and subsample a number (the batchsize) of rows of X at each iteration.

Comment. In §3, we showed that the key difference between `Signum` and `Adam` lies in the variance correction term $\beta\texttt{EMA}_\beta[(m_{k-1} - g_k)^2]$ in the denominator. Understanding how this term evolves is essential: it cannot be approximated by a constant. In the second row of Figure 15, we observe that the variance estimate not only varies over time, but also differs in scale across the three blocks—mimicking the parameter groupings in transformer models. This block-wise variation reinforces the idea that the variance term dynamically adapts to the local curvature and cannot be substituted by a fixed value. In Figure 16 and 13, we show a similar effect in heterogeneous quadratic and language models, respectively: replacing $\beta\texttt{EMA}_\beta[(m_{k-1} - g_k)^2]$ with a fixed constant ϵ cannot provide the same adaptive effect.

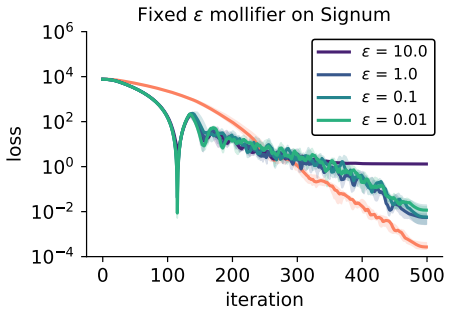

Figure 16: *Counterpart of Figure 13 for the heterogeneous quadratic example. We do not observe gains adding a fixed mollifier $m_k/\sqrt{m_k^2 + \epsilon}$ (placing inside or outside $\sqrt{\cdot}$ has no qualitative effect).*

Appendix F. Signal Processing Perspective

In this last section, we examine `Adam` through a signal processing lens, to get qualitative insights into its distinction with `Signum` and other `SignSGD` with momentum variants. Setting $\beta_1 = \beta_2 = \beta$,

we can write the Adam update, without bias correction (see §C.5) as simply

$$d_k = \sqrt{1-\beta}\frac{\sum_{i=0}^{k}\beta^i g_{k-i}}{\sqrt{\sum_{i=0}^{k}\beta^i g_{k-i}^2}}.$$

where $(g_k)_k$ is the gradient signal. The results in Figure 2 clearly show that this is a near-optimal performing method.

One might wonder if this special case allows for a simpler graphical interpretatoin of Adam.

Graphical intuition. We denote by d_k the update of Adam:

$$d_k = \left(\sqrt{\texttt{EMA}_\beta[g_k^2]} + \epsilon\right)^{-1} \texttt{EMA}_\beta[g_k]$$

and plot its dynamics as a function of a synthetic one-dimensional gradient in Figure 17.

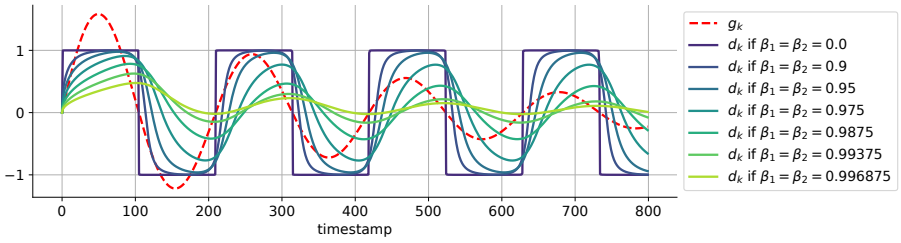

Figure 17: Filtering effect for same $\beta_1 = \beta_2$.

In the example of Figure 17, we chose the synthetic gradient signal $g_k = 1.8\sin(0.03k)\exp(-0.0025k)$: this is a damped periodic signal. Note that this is pure filtering, there is no loss or learning process. We note the following:

1. $\beta_1 = \beta_2 = 0$ is obviously just $\texttt{sign}(g_k)$. This is plotted for comparison.
2. For any $\beta_1 = \beta_2 \neq 0$, d_k is bounded by 1 in magnitude. It's dynamics however, for e.g. $\beta_1 = \beta_2 > 0$ is smooth and follows more closely the gradient, while being bounded. It is somehow a rescaled version.
3. Very interestingly, d_k is blind to the decay term $\exp(-0.0025k)$, the output is perfectly periodic for every $\beta_1 = \beta_2$.

Towards understanding what is a functional form for this, note that d_k **cannot be reduced to momentum on the sign or sign on the momentum(Signum)**: both variants actually destroy the signal shape, while d_k maintains the shape of the original signal and has clear invariance properties. The behavior of signSGD with momentum (2 variants) is shown in Figure 18: as one can see, the behavior is drastically different from d_k in Figure 17.

Properties. Adam can be seen as a very special operator T on gradient sequences $(g_k)_{k=0}^{\infty} \in \mathcal{G} \subseteq \ell_\infty$ (with normed vector space structure and notation), indeed has four distinctive properties. $T : (g_k)_{k=0}^{\infty} \to (d_k)_{k=0}^{\infty}$.

1. It is **causal**.
2. It is **invariant to positive scaling**: $T(\alpha \cdot g) = T(g)$, for any $\alpha > 0$.
3. It is **odd**: $T(-g) = -T(g)$.
4. It has **bounded** infinity norm: $\|T(g)\|_\infty \leq 1$ for all $g \in \ell_\infty$.
5. **Density**: For any $b \in [-1, 1]$ and any arbitrary $k > 0$, there exists $(g_k)_{k=0}^{\infty}$ such that $d_k = b$.

We are amazed by these rich set of properties, thickening our interest in better understanding the properties of `Adam` mollification, which we study in §3.

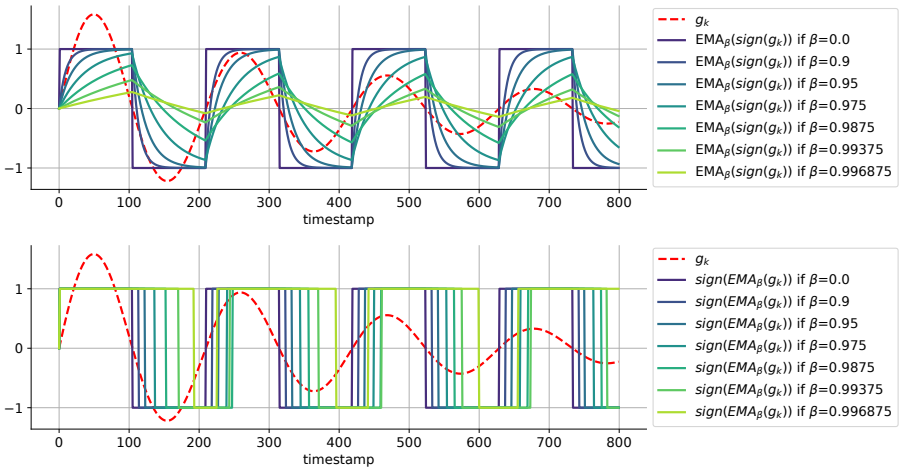

Figure 18: Filtering induced by signSGD with momentum (2 variants, the one below is `Signum`). Compare with Figure 17.

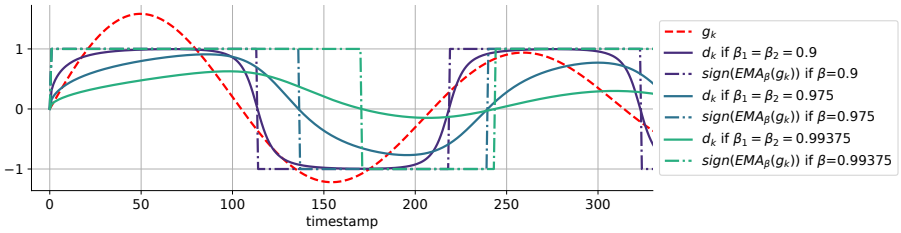

Figure 19: `Adam`-like filtering compared to sign of EMA (`Signum`), detail.