OPT2024: 16th Annual Workshop on Optimization for Machine Learning

# AdEMAMix: Better and Faster Training with Older Gradients

**Matteo Pagliardini**[*] MATTEO.PAGLIARDINI@EPFL.CH
*EPFL*

**Pierre Ablin** PABLIN@APPLE.COM
*Apple*

**David Grangier** D_GRANGIER@APPLE.COM
*Apple*

## Abstract

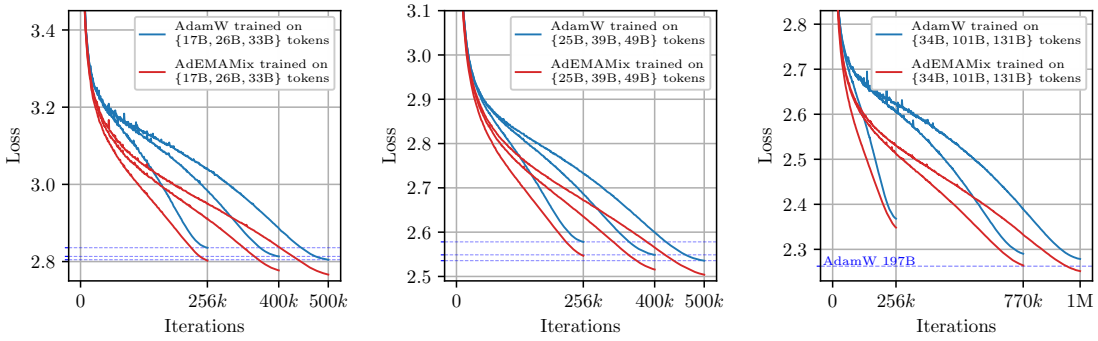
Momentum based optimizers are central to a wide range of machine learning applications. These typically rely on an Exponential Moving Average (EMA) of gradients, which decays exponentially the present contribution of older gradients. This accounts for gradients being local linear approximations which lose their relevance as the iterate moves along the loss landscape. This work questions the use of a single EMA to accumulate past gradients and empirically demonstrates how this choice can be sub-optimal: a single EMA cannot simultaneously give a high weight to the immediate past, and a non-negligible weight to older gradients. Building on this observation, we propose AdEMAMix, a simple modification of the Adam optimizer with a mixture of two EMAs to better take advantage of past gradients. Our experiments on language modeling show—quite surprisingly—that gradients can stay relevant for tens of thousands of steps. They help to converge faster, and often to lower minima: e.g., a $1.3$B parameter AdEMAMix LLM trained on $101$B tokens performs comparably to an AdamW model trained on $197$B tokens ($+95\%$). Moreover, our method significantly slows-down model forgetting during training. Our work motivates further exploration of different types of functions to leverage past gradients, beyond EMAs. For an extended version of this work, see: https://arxiv.org/abs/2409.03137.

## 1. Introduction

With large neural networks, deep-learning has revolutionized numerous fields, such as computer vision and natural language processing. At the heart of this paradigm lies the challenge of optimizing complex, non-convex loss functions using noisy gradient estimates. This optimization process is typically carried out using variants of Stochastic Gradient Descent (SGD) [28] or adaptive methods such as Adam and AdamW [16, 19], which have become ubiquitous in training state-of-the-art models [1, 5–8, 27, 33, 36].

A key component in many of these iterative optimization algorithms is momentum, which has long been shown to accelerate convergence [21] and often leads to solutions with superior generalization properties [31]. By accumulating gradient vectors over successive optimization steps, momentum helps overcome small local variations of the loss landscape, potentially escaping shallow local minima, and accelerate in plateau regions [12, 26, 29]. Both SGD with momentum (SGD+M) and Adam incorporate momentum under the form of Exponential Moving Averages (EMAs) of past

[*] Work done while interning at Apple.




(*a*) 110M parameters. (*b*) 330M parameters. (*c*) 1.3B parameters.

Figure 1: **Comparing AdamW and AdEMAMix on language modeling.** In **(a,b,c)**, we plot the loss obtained using AdamW and AdEMAMix (our optimizer) to train Transformer models of various sizes on the Redpajama dataset. In **(a)**, we train multiple baselines for $256k$, $400k$, and $500k$ iterations, resulting in processing from 17B to 33B tokens. Two AdamW runs with different number of iterations look very different as we use a cosine-decay for the learning rate. We compare those baselines to training AdEMAMix for $256k$ iterations. We observe that our method reaches a similar loss as an AdamW model trained on nearly twice the number of tokens. Analogous comparisons can be derived from **(b)** and **(c)**. Notably, in **(c)**, a 1.3B parameter AdEMAMix model trained on 101B tokens performs comparably to an AdamW model trained on 197B tokens (95% more, blue horizontal line).

gradients $\mathcal{G}^T = \{\boldsymbol{g}^{(0)}, \dots, \boldsymbol{g}^{(T)}\}$:

$$\text{EMA}(\beta, \mathcal{G}^T) \triangleq \beta \cdot \text{EMA}(\beta, \mathcal{G}^{(T-1)}) + (1-\beta)\boldsymbol{g}^{(T)} = \sum_{i=0}^{T} \beta^i (1-\beta)\boldsymbol{g}^{(T-i)}. \tag{EMA}$$

Two considerations support the use of EMAs. From a practical standpoint, the recursive formula of EMA allows for efficient implementations, which do not require maintaining a buffer of past gradients. From a theoretical standpoint, gradient descent with momentum leads to optimal convergence rates for quadratics [22, 25]. However, those results do not guarantee any optimality for general non-quadratic cases [13].

The use of momentum in optimization is grounded in the varying nature of gradients. As local linear approximations of the loss landscape, their information can quickly become outdated as the optimization process progresses [24]. For this reason, practitioners typically employ moderate momentum values (i.e. $\beta \approx 0.8$ or $0.9$), effectively creating a moving average of recent gradients while discarding older information. Selecting larger $\beta$ values seems counter-intuitive, as it would suggest that older gradients maintain their relevance over extended periods of training. While it is tempting to see the use of small $\beta$s as a confirmation of the limited temporal relevance of gradients, our work reveals instead that older gradients can efficiently be used. When we increase $\beta$, we decrease the relative importance of recent gradients, and the iterate now fails to respond to local changes in the loss landscape. We observe that a single EMA cannot both give a significant weight to recent gradients, and give a non-negligible weight to older gradients. However, a linear combination between a "fast-changing" (e.g. $\beta = 0.9$ or $\beta = 0$) and a "slow-changing" (e.g. $\beta = 0.9999$) EMA

allows the iterate to beneficiate from (i) the great speedup provided by the larger (slow-changing) momentum, while (ii) still being reactive to small changes in the loss landscape (fast-changing). More precisely, we find the following statement to convey an important intuition behind this approach:

> *While changing the direction of the slow momentum is difficult, any adjustment orthogonal to that direction is easy—which favors fast progress in sinuous canyon-like landscapes.*

A toy illustration of this can be seen in Fig. 3. Based on this idea, we propose AdEMAMix (**Ad**aptive **EMA Mix**ture), a novel Adam based optimizer which successfully leverages very old gradients to reach better solutions.

**Contributions.** Our contributions can be summarized as follows: **(i)** We propose AdEMAMix, a novel optimizer which better leverages past gradients by avoiding a common pitfall of EMA-based optimizers (see § 3). **(ii)** We empirically demonstrate the superiority of our method over Adam by training Transformer language models of up to 1.3B parameters (see § 4). **(iii)** We show AdEMAMix forgets the training data slower when compared to Adam (see Fig. 2). **(iv)** More broadly, our findings contribute to a deeper understanding of the optimal balance between using historical gradients and adapting to the rapidly changing loss landscape. Our work invites further research in methods combining old and recent gradients, beyond EMAs.

## 2. Related Work

**Works on understanding momentum.** From the seminal work of [25], many publications analyzed the effect of gradient descent + momentum (GD+M) in both convex and non-convex settings [4, 9, 10, 15, 30]. While the acceleration in the noise-free setting has been long theorized for convex functions, several publications indicate this effect might not necessarily extend to stochastic settings [15, 17, 35], emphasizing instead a link between momentum and effective learning rate. Recent work have been seeking to understand the impact of momentum on generalization through studying the implicit bias of momentum methods [11, 23], exposing a preference of SGD+M for lower norm solutions. Those further exposed a link between higher momentum and higher effective learning rate and higher variance reduction. Despite the volume of prior work on the subject, our understanding of momentum methods in non-convex stochastic settings is still incomplete [35]. Oscillatory behaviours, and the sometimes ambiguous effect of variance on optimization render the analysis tedious. From a theoretical standpoint, our work raises several questions. First, given that we gain from averaging very old gradients, what can it reveal of the loss landscape and the consistency of one batch's gradient during training? Second, would our approach not decrease the variance up to a point that is harming generalization [11]? While no answer to those questions is given in this work, we provide a toy justification which indicates that large momentums can have a positive impact in noise-free non-convex settings (see Fig. 3)—indicating the improvement of our approach is at least partially explainable without considering variance-reduction effects. We moreover expose a link between momentum and forgetting the training data (see Fig. 2), which to our knowledge is novel.

**Works on deep-learning optimizers.** Despite the popularity of Adam and AdamW [16, 19] in training deep neural networks, optimizer design is a rich field of research and we focus on a few of the works most relevant to this study. Chen et al. [2] use algorithm discovery to derive the Lion optimizer. Contrary to Adam, Lion uses a single momentum term and the sign function to produce updates with the same magnitude across dimensions. Interestingly, Chen et al. [2] also report better scores are obtained when using a slightly larger momentum term ($\beta = 0.99$). In this work we show how increasing the momentum well beyond this value can still be beneficial. Recently,

Liu et al. [18] introduced Sophia, a scalable second-order optimizer designed for LLM training. Sophia uses a Hessian-based pre-conditioner which better normalizes the step size, penalizing steps in high curvature direction and accelerating in low curvature directions. Understanding in which circumstances those novel optimizers bring improvements is still being investigated [14], and Adam's dominance remains mostly unchallenged. Most relevant to us, Lucas et al. [20, AggMo] also observe that using a combination of EMAs can enable the use of larger $\beta$s, and incorporates a sum of $K$ momentum terms into *GD* . They show their approach reaches similar performances as baseline optimizers, with a faster convergence. In contrast, we modify *Adam*, which is the workhorse algorithm for large-scale optimization, and introduce schedulers that are critical to reaching good performances at larger scales. As a result, we not only converge faster, but better, and outperform Adam by a significant margin. Finally, Szegedy et al. [32] propose a general framework to derive and study optimizers with linear combinations of memory vectors—which encompasses EMA mixtures.

## 3. Our method: AdEMAMix

**Setup & notations.** Let $\mathcal{L}_{\boldsymbol{\theta}} : \mathcal{X} \mapsto \mathbb{R}$ be a loss function parameterized by $\boldsymbol{\theta}$, and mapping inputs $\boldsymbol{x} \in \mathcal{X}$ to $\mathbb{R}$. Given a sampled batch $\boldsymbol{x}$, let $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{x})$ be a stochastic gradient of the loss w.r.t. $\boldsymbol{\theta}$. To minimize the empirical loss, the Adam optimizer [16] relies on first and second moments, resp. $\boldsymbol{m}$ and $\boldsymbol{\nu}$, estimated via two EMAs parametrized by $(\beta_1, \beta_2) \in [0, 1[^2$. A weight-decay parameter $\lambda \in \mathbb{R}^+$ is often used as in Loshchilov and Hutter [19]:

$$\begin{cases} \boldsymbol{m}^{(t)} = \beta_1 \boldsymbol{m}^{(t-1)} + (1-\beta_1)\boldsymbol{g}^{(t)}, & \hat{\boldsymbol{m}}^{(t)} = \frac{\boldsymbol{m}^{(t)}}{1-\beta_1^t} \\ \boldsymbol{\nu}^{(t)} = \beta_2 \boldsymbol{\nu}^{(t-1)} + (1-\beta_2){\boldsymbol{g}^{(t)}}^2, & \hat{\boldsymbol{\nu}}^{(t)} = \frac{\boldsymbol{\nu}^{(t)}}{1-\beta_2^t} \\ \boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} - \eta\big(\frac{\hat{\boldsymbol{m}}^{(t)}}{\sqrt{\hat{\boldsymbol{\nu}}^{(t)}}+\epsilon} + \lambda \boldsymbol{\theta}^{(t-1)}\big). \end{cases} \tag{AdamW}$$

With $t > 0$ being the timestep, $\eta$ being the learning rate, and $\epsilon$ a small constant. Initially $\boldsymbol{m}^{(t=0)} = \boldsymbol{\nu}^{(t=0)} = \mathbf{0}$. To prevent the bias induced by the initial $\boldsymbol{m}^{(t=0)}$ and $\boldsymbol{\nu}^{(t=0)}$, the outputs of the two EMAs are corrected into $\hat{\boldsymbol{m}}^{(t)}$ and $\hat{\boldsymbol{\nu}}^{(t)}$. Those are used to compute the final weight update, scaled by the learning rate.

**How far to look into the past?** A typical value for $\beta_1$ is $0.9$. The larger the $\beta$, the more uniform the average is. To put this in perspective—observing that $\sum_{i=0}^{\infty} \beta^i(1-\beta) = 1$ for $\beta \in [0, 1[$—the number of successive previous steps receiving a cumulative weight of $0.5$, is $t_{half} = \frac{\ln(0.5)}{\ln(\beta)} - 1$. For $\beta = 0.9$, $t_{half} \approx 6$, meaning that half of the weight is given to the previous six gradients. This observation can also be extended to SGD with e.g. polyak or nesterov momentums [22, 25], which typically relies on similar $\beta$ values. The value of $\beta_1$ is rarely increased beyond $\sim 0.9$. In our experiments with AdamW, increasing $\beta_1$ further degraded the performance. Does this mean older gradients are outdated? We show that this is not the case, rather, increasing beta is reducing the sensitivity to recent gradients too much. We design AdEMAMix such that the sensitivity to recent gradients is kept, while also incorporating information from much older gradients using an additional momentum term. This allows for the use of much larger $\beta$ values e.g. $0.9999$. To compare, for $\beta = 0.9999$, $t_{half} \approx 6{,}930$, spreading half of the mass over the previous $6{,}930$ past gradients.

**AdEMAMix.** To keep a high sensitivity to recent gradients, while also incorporating information from older gradients, we add a second EMA (changes compared to AdamW are in Blue):

$$\begin{cases} \boldsymbol{m}_1^{(t)} = \beta_1 \boldsymbol{m}_1^{(t-1)} + (1-\beta_1)\boldsymbol{g}^{(t)}, & \hat{\boldsymbol{m}}_1^{(t)} = \frac{\boldsymbol{m}_1^{(t)}}{1-\beta_1^t} \\ \boldsymbol{m}_2^{(t)} = \beta_3 \boldsymbol{m}_2^{(t-1)} + (1-\beta_3)\boldsymbol{g}^{(t)} & \\ \boldsymbol{\nu}^{(t)} = \beta_2 \boldsymbol{\nu}^{(t-1)} + (1-\beta_2){\boldsymbol{g}^{(t)}}^2, & \hat{\boldsymbol{\nu}}^{(t)} = \frac{\boldsymbol{\nu}^{(t)}}{1-\beta_2^t} \\ \boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} - \eta\big(\frac{\hat{\boldsymbol{m}}_1^{(t)} + \alpha \boldsymbol{m}_2^{(t)}}{\sqrt{\hat{\boldsymbol{\nu}}^{(t)}}+\epsilon} + \lambda \boldsymbol{\theta}^{(t-1)}\big). \end{cases} \quad \text{(AdEMAMix)}$$

In our experiments, while the values of $\beta_1, \beta_2$ remain similar to those of equation AdamW, we often use $\beta_3 = 0.9999$. We find $\alpha \in [4, 10]$ to work well in practice.

**Tackling early training instabilities.** Early training instabilities are commonplace when training deep models, and empirically motivated the introduction of methods such as learning rate warmup and gradient clipping. While we use learning rate warmup in all our experiments, we still noticed AdEMAMix models using a large $\beta_3$ would diverge early. This, despite not using bias correction over $\boldsymbol{m}_2$, which lets the momentum buffer fill itself slowly. Those failed runs are characterized by updates of large magnitudes in the early phase of training. For this reason, we progressively increase the values of $\beta_3$ and $\alpha$ using schedulers. For $\alpha$ we use a linear scheduler. A linear scheduler for $\beta_3$ would be ill-fitted as the same increment of $\beta_3$ have a different impact for different values of $\beta_3$. For instance, observe that an increase of $\beta$ of $\delta_\beta = 0.0001$ barely increases the $t_{half}$ for $\beta = 0.9$, while $0.999 \rightarrow 0.999 + \delta_\beta$ increases the $t_{half}$ of 77. For this reason, we design the $\beta_3$ scheduler to *increase* $t_{half}$ *linearly*. The two schedulers are summarized below:

$$\alpha^{(t)} = f_\alpha(t, \alpha, T_\alpha) = \min(\frac{t\alpha}{T_\alpha}, \alpha), \quad (f_\alpha)$$

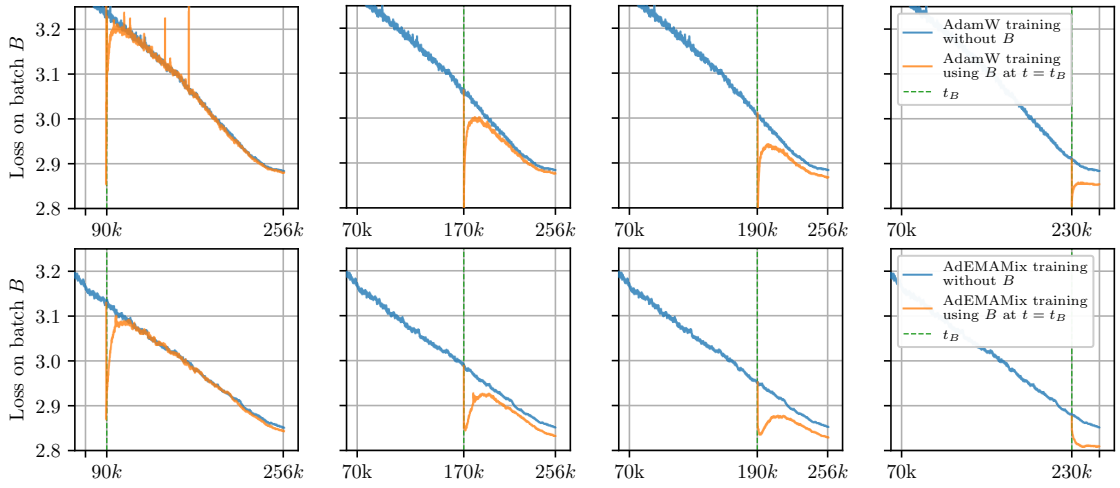
$$\beta_3^{(t)} = f_{\beta_3}(t, \beta_3, \beta_{start}, T_{\beta_3}) = \min\big(\exp\big(\frac{\ln(\beta_{start})\ln(\beta_3)}{(1-\frac{t}{T_{\beta_3}})\ln(\beta_3) + \frac{t}{T_{\beta_3}}\ln(\beta_{start})}\big), \beta_3\big). \quad (f_{\beta_3})$$

With $T_\alpha$ and $T_{\beta_3}$ are resp. the warmup times for $\alpha^{(t)}$ and $\beta_3^{(t)}$ to reach their final and maximal values. In practice we always set those two to the same value: $T_\alpha = T_{\beta_3} = T_{\alpha,\beta_3}$, and we typically use $T_{\alpha,\beta_3} = T$, with $T$ being the total number of iterations. $\beta_{start}$ is always set to $\beta_1$ in our experiments.

**Hyperparameter sensitivity.** While we introduce up to four new hyperparameters: $\alpha$, $\beta_3$, $T_\alpha$, and $T_{\beta_3}$. In practice we always set $T_\alpha = T_{\beta_3} = T_{\alpha,\beta_3}$, and use $T_{\alpha,\beta_3} = T$ in most cases. While all of our experiments on language modeling use $\beta_3 = 0.9999$, other values such as $0.999$ or even $0.99999$ still can outperform the AdamW baseline. Overall, we find the ranges of values of $\alpha$, $\beta_3$ and $T_{\alpha,\beta_3}$ providing improvements over AdamW to be wide.

## 4. Results

**Experimental setup.** We use a transformer architecture [34]. Our experiments use sequences of 1,024 tokens. We experiment with three model sizes: 110M, 335M, and 1.3B. We use 3k warmup steps followed by a cosine decay. We extensively tuned the hyperparameters for both AdamW and AdEMAMix models. We use the RedPajama v2 [3] dataset for all of our experiments. We use batch sizes of 64, 96 and 128 for respectively our 110M, 335M, and 1.3B parameter models. For AdEMAMix, we use $\beta_3 = 0.9999$ and $\alpha \in \{5, 8, 10\}$ depending on the model.




(*a*) $t_B = 90k$. (*b*) $t_B = 170k$. (*c*) $t_B = 190k$. (*d*) $t_B = 230k$.

Figure 2: **Measuring forgetting using a held-out batch $B$.** The top row is for AdamW, the bottom row is for AdEMAMix. We trained one AdamW and AdEMAMix model on a RedPajama dataset *not* containing the batch $B$, those runs are in blue. We then run multiple experiments where we inject $B$ in the training data at a specific timestep $t_B$. Those runs are in orange. To inspect how much influence $B$ had when it is injected at $t_B$, we can observe the evolution of the gap between the blue and the orange curves. For both optimizers, we observe a rapid decrease of the loss on $B$ right after training on $B$. The sharpness of this decrease in loss is more pronounced for AdamW compared to AdEMAMix. However, when using AdamW, the loss on $B$ then increases faster, which we interpret as the model forgetting $B$ faster. In contrast, the curves for AdEMAMix are smoother, the loss on $B$ goes back up slower, and ultimately $B$ had a bigger impact on the training when using AdEMAMix—as can be seen by looking at the larger gap between the orange and blue curves for the last iteration. Finally, the forgetting behaviour evolve during training, with the later training batches being remembered better.

**Why not simply increasing AdamW's $\beta_1$?** We train multiple 110M models using Adam with large $\beta_1 \in \{0.99, 0.999, 0.9999, 0.99999\}$. When we use a large $\beta_1$ from the beginning of training, we observe instabilities for larger $\beta_1$ values and no $\beta_1 > 0.9$ improves upon the AdamW baseline. One can imagine this to be due to increasing $\beta_1$ too early. Therefore, we also modify AdamW and add the same scheduler on $\beta_1$ as we use on AdEMAMix's $\beta_3$. $\beta_1$ is now increased steadily over the entire training duration. While this mostly stabilizes the training, none of the experiments outperformed the baseline using $\beta_1 = 0.9$. Those experiments show that simply increasing the $\beta_1$ value in AdamW is not enough, which justifies our design of AdEMAMix.

**Better perplexity for the same number of steps.** For all model sizes, AdEMAMix always outperforms the AdamW baseline. In Fig. 1, we show the validation loss curves for AdamW and AdEMAMix models trained on various numbers of tokens. For 110M parameter models, training for $256k$ iterations gives similar results as training an AdamW model for $500k$ iterations. For 1.3B parameter models, training using $770k$ steps is on par with training the baseline for 1.5M iterations.

**AdEMAMix models forget the training data slower.** When comparing the forgetting curves for AdamW and AdEMAMix in Fig. 2, we see striking differences. AdamW models forget much faster—the loss over $B$ increases faster—than AdEMAMix models. Moreover, at the end of training, batches processed by AdEMAMix see their loss being improved over many thousands of iterations.

## 5. Conclusion

In this work, we propose a novel optimizer which combines two momentum terms. A slow (large $\beta$) momentum gathers information over many timestep, while a fast (slow $\beta$) momentum can adapt the trajectory of the iterates to the rapidly changing loss landscape. We demonstrate the superiority of our optimizer over AdamW through a set of experiments on text modeling. We moreover reveal how our optimizer forgets the training data at a slower pace.

## References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

[2] 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 *NeurIPS*, 2023.

[3] Together Computer. Redpajama: an open dataset for training large language models, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

[4] Aaron Defazio. Understanding the role of momentum in non-convex optimization: Practical insights from a lyapunov analysis. *CoRR*, abs/2010.00406, 2020. URL https://arxiv.org/abs/2010.00406.

[5] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme Ruiz, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd van Steenkiste, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Collier, Alexey A. Gritsenko, Vighnesh Birodkar, Cristina Nader Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetic, Dustin Tran, Thomas Kipf, Mario Lucic, Xiaohua Zhai, Daniel Keysers, Jeremiah J. Harmsen, and Neil Houlsby. Scaling vision transformers to 22 billion parameters. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 7480–7512. PMLR, 2023.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*. OpenReview.net, 2021.

[8] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, and Kevin Stone. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[9] Nicolas Flammarion and Francis R. Bach. From averaging to acceleration, there is only a step-size. In *COLT*, volume 40 of *JMLR Workshop and Conference Proceedings*, pages 658–695. JMLR.org, 2015.

[10] Euhanna Ghadimi, Hamid Reza Feyzmahdavian, and Mikael Johansson. Global convergence of the heavy-ball method for convex optimization. In *ECC*, pages 310–315. IEEE, 2015.

[11] Avrajit Ghosh, He Lyu, Xitong Zhang, and Rongrong Wang. Implicit regularization in heavy-ball momentum accelerated stochastic gradient descent. In *ICLR*. OpenReview.net, 2023.

[12] Gabriel Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL http://distill.pub/2017/momentum.

[13] Baptiste Goujaud, Adrien Taylor, and Aymeric Dieuleveut. Provable non-accelerations of the heavy-ball method. *arXiv preprint arXiv:2307.11291*, 2023.

[14] 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. In *NeurIPS*, 2023.

[15] Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham M. Kakade. On the insufficiency of existing momentum schemes for stochastic optimization. In *ICLR*. OpenReview.net, 2018.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[17] Guillaume Leclerc and Aleksander Madry. The two regimes of deep network training. *CoRR*, abs/2002.10376, 2020.

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

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR (Poster)*. OpenReview.net, 2019.

[20] James Lucas, Shengyang Sun, Richard S. Zemel, and Roger B. Grosse. Aggregated momentum: Stability through passive damping. In *ICLR (Poster)*. OpenReview.net, 2019.

[21] Arkaddii S Nemirovskii and Yu E Nesterov. Optimal methods of smooth convex minimization. *USSR Computational Mathematics and Mathematical Physics*, 25(2):21–30, 1985.

[22] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. In *Doklady Akademii Nauk SSSR*, 1983. URL https://api.semanticscholar.org/CorpusID:202149403.

[23] Hristo Papazov, Scott Pesme, and Nicolas Flammarion. Leveraging continuous time to understand momentum when training diagonal linear networks. In *AISTATS*, volume 238 of *Proceedings of Machine Learning Research*, pages 3556–3564. PMLR, 2024.

[24] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org, 2013.

[25] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 3(4):864–878, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(63)90382-3. URL https://www.sciencedirect.com/science/article/pii/0041555363903823.

[26] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL https://www.sciencedirect.com/science/article/pii/S0893608098001166.

[27] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

[28] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[29] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[30] Othmane Sebbouh, Robert M. Gower, and Aaron Defazio. Almost sure convergence rates for stochastic gradient descent and stochastic heavy ball. In *COLT*, volume 134 of *Proceedings of Machine Learning Research*, pages 3935–3971. PMLR, 2021.

[31] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/sutskever13.html.

[32] Balázs Szegedy, Domonkos Czifra, and Péter Kőrösi-Szabó. Dynamic memory based adaptive optimization, 2024. URL https://arxiv.org/abs/2402.15262.

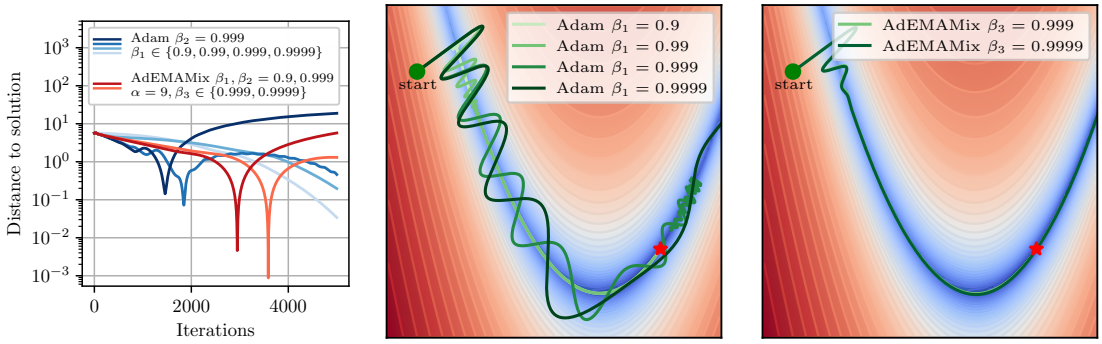
[33] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[35] Kun Yuan, Bicheng Ying, and Ali H. Sayed. On the influence of momentum acceleration on online learning. *Journal of Machine Learning Research*, 17(192):1–66, 2016. URL http://jmlr.org/papers/v17/16-157.html.

[36] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *CVPR*, pages 1204–1213. IEEE, 2022.

## Appendix A. Toy example




(*a*) $\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^\star\|_2$. (*b*) Adam trajectories. (*c*) AdEMAMix trajectories.

Figure 3: **Comparing Adam and AdEMAMix on the Rosenbrock function.** Starting from $\boldsymbol{x}^{(0)} = [-3, 5]$, we minimize the Rosenbrock function $f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$. The global minimum ($\star$) is $\boldsymbol{x}^\star = [1, 1]$. We use $\beta_2 = 0.999$ for Adam and $(\beta_1, \beta_2, \alpha) = (0.9, 0.999, 9)$ for AdEMAMix (see § 3). We reduce the learning rate for AdEMAMix to compensate for the influence of $\alpha$. We do a sweep over $\beta_1$ (resp. $\beta_3$) for Adam (resp. for AdEMAMix). In **(b)**, When Adam's $\beta_1$ is small (e.g. $0.9$), the iterates do not oscillate but convergence is slow. Increasing $\beta_1$ makes the iterates move faster but with large oscillations. In contrast, for AdEMAMix in **(c)**, we observe that despite $\beta_3$ being large, the iterates moves fast and without oscillations. This results in reaching better solutions faster as can be seen in **(a)**.

## Appendix B. Deriving the $\beta_3$ scheduler

Let's consider $S(t)$, the sum of the weights given to the last $t$ gradients by an EMA parameterized by $\beta \in [0, 1[$:

$$S(t) = (1 - \beta) \sum_{i=0}^{t} \beta^i$$

We want to know which timestep $t$ would correspond to a cumulative weight of $0.5$:

$$(1 - \beta) \sum_{i=0}^{t} \beta^i = 0.5 \Leftrightarrow \beta^{t+1} = 0.5 \Leftrightarrow t = \frac{\ln(0.5)}{\ln(\beta)} - 1$$

Let $f(\beta) = \frac{\ln(0.5)}{\ln(\beta)} - 1$. This function provides how many past consecutive gradients receive a cumulative weight of $0.5$.

Its inverse is:

$$f^{-1}(t) = 0.5^{\frac{1}{t+1}}$$

We want a scheduler which increases $\beta$ from $\beta_{start}$ to $\beta_{end}$ such that $f(\beta)$ increases linearly. Given an interpolating parameter $\mu \in [0, 1]$, this scheduler can be written as:

$$\beta(\mu) = f^{-1}((1 - \mu) f(\beta_{start}) + \mu f(\beta_{end}))$$

By replacing $f$ and $f^{-1}$ by their respective formula, one can arrive to:

$$\beta(\mu) = \exp\left(\frac{\ln(\beta_{start})\ln(\beta_{end})}{(1-\mu)\ln(\beta_{end}) + \mu\ln(\beta_{start})}\right)$$

By setting $\beta_{end} = \beta_3$ and $\mu = \frac{t}{T_{\beta_3}}$, we arrive to the $\beta_3$-scheduler introduced in § 3. We show the shape of our scheduler and compare it to a linear scheduler in Fig. 4.

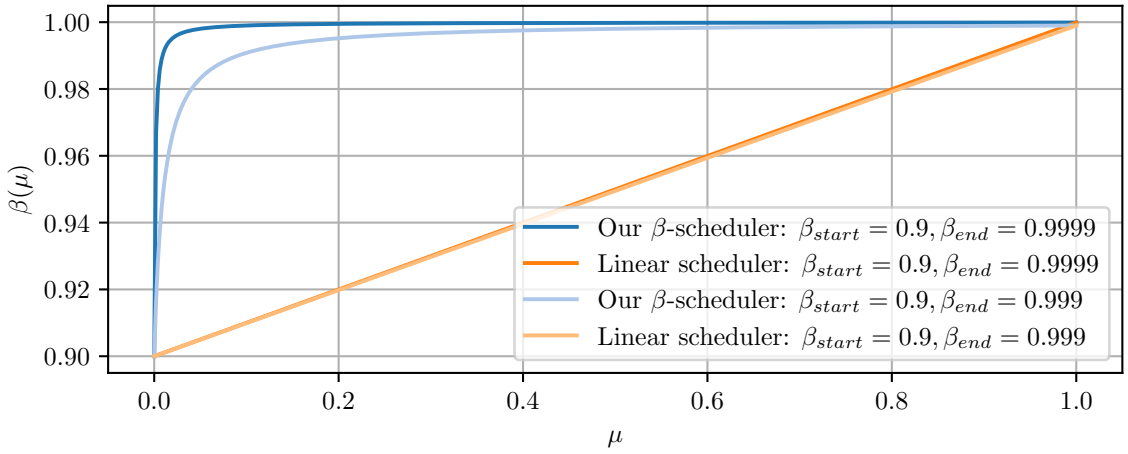



Figure 4: **AdEMAMix's $\beta_3$ scheduler.** We compare our scheduler to a linear scheduler for $\beta_{start} = 0.9$ and $\beta_{end} \in \{0.999, 0.9999\}$. While our scheduler looks more aggressive at first glance, it increases fast for smaller values of $\beta$, and slowly for larger ones associated. This makes sense as the same increase of $\beta$ for larger $\beta$ values has a bigger impact than that same increase applied to a smaller value of $\beta$. The two linear schedulers look practically the same, despite values of $\beta_{end}$ differing by one order of magnitude. This is not the case with our scheduler.