HiLD 2024: 2nd Workshop on High-dimensional Learning Dynamics

Analyzing & Eliminating Learning Rate Warmup in GPT Pre-Training

Atli Kosson ATLI.KOSSON@EPFL.CH
Bettina Messmer BETTINA.MESSMER@EPFL.CH
Martin Jaggi MARTIN.JAGGI@EPFL.CH
EPFL, Switzerland

Abstract

Learning Rate Warmup is a popular heuristic for training neural networks, which downscales early updates relative to later ones. This aids training, suggesting that the initial updates are too large in some sense, but *why and by which criteria* remains unclear. In this work we explore this for small GPT training by assessing and controlling the update size via various metrics. We find the standard ℓ_2-norm of the updates to be insufficient, but using relative changes of either the matrix weights or neural representations is promising for reducing or eliminating the need for explicit warmup. Quantifying the updates in representation space in particular can help withstand changes in the gradient signal-to-noise ratio or "critical batch size" throughout training, which warmup can help counteract but simpler weight based methods fail to account for.

1. Introduction

Neural networks are typically trained using variations of stochastic gradient descent, where the learning rate hyperparameter scales the size of weight updates. Throughout training, the learning rate is often adjusted according to a learning rate schedule. This schedule frequently includes a warmup phase, where the learning rate starts low and is increased to a target value before being reduced according to a decay schedule. Both the choice of warmup and decay strategy can significantly affect the final model performance. In this work, we focus on the linear warmup introduced by Goyal et al. [8] for large batch size ResNet [9] training, which is also commonly used for transformers [33].

The length of the warmup is a hyperparameter that requires tuning, which is complicated by the fact that the reasons for its effectiveness are somewhat unclear. Empirically, warmup helps stabilize training and allows for larger learning rates throughout the rest of training, which can speed up the process and provide beneficial regularization [8]. By definition, warmup must achieve this by decreasing the size of early updates, but why does this help? *Are the initial updates too large for some reason? How should we quantify large updates?*

This work explores warmup from this perspective, focusing on GPT2 [27] training with adaptive optimizers like AdamW [22] and Lion [3]. We identify three key issues that necessitate warmup:

1. The way Adam handles momentum can lead to artificially large initial updates.
2. Early optimizer updates are not proportionate to the initialization magnitude of matrices.
3. The gradients of early samples are highly correlated, limiting effective mini-batch sizes.

We demonstrate that simple modifications to the optimizer, eliminating momentum bias correction in AdamW and scaling matrix updates similarly to Rotational Optimizers [17], can mitigate the first two issues. For the third issue, we analyze changes to the internal neural representations of the network. When gradients of different samples are highly correlated, the internal representations change rapidly,

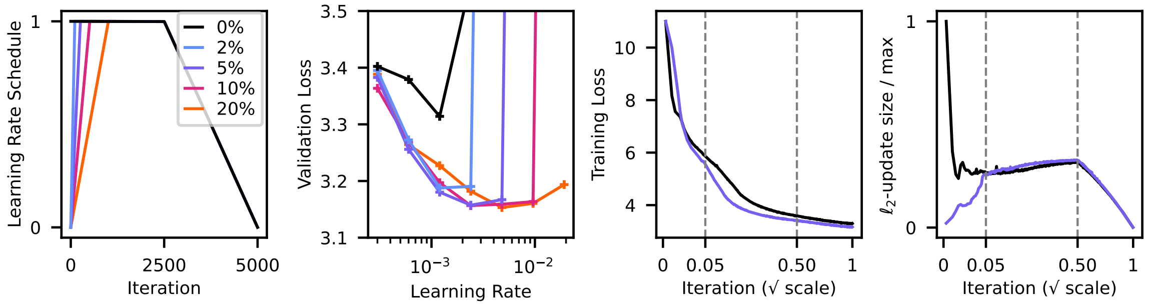

Figure 1: Warmup significantly benefits GPT2 training with AdamW. **Panel 1:** Trapezoidal learning rate schedules with different warmup lengths and 50% linear cooldown. **Panel 2:** Final validation loss for various learning rate and warmup configurations. Note the performance gap between no-warmup (black) and other configurations. **Panel 3:** Training curves comparing the best no-warmup run to a 5% warmup with the same learning rate. The warmup run quickly surpasses the no-warmup run. **Panel 4:** Comparison of ℓ_2-update norms for these runs shows large initial updates without warmup.

which we conjecture can lead to issues with the non-linearities of the network. This is analogous to having a low *critical batch size* [26] early in training, preventing the use of the peak learning rate. We derive a scaling factor based on the signal-to-noise ratio of the gradient to mitigate this, functioning as an automatic learning rate warmup. Alternatively, using high momentum values with the first two methods can enable performant training without warmup in our setting.

2. Baseline Experimental Setup & Results

Our main experiments focus on the training of a 124M parameter GPT2 [27] model. The model has 12 transformer blocks with an embedding dimension of 768. Our base training is performed at batch size 480 with a sequence length of 1024. We train for 5000 iterations which translates into roughly 20 tokens per parameter, as suggested by Chinchila [10]. The baselines use AdamW [22] (see algo. 1) with weight decay $\lambda = 0.1$, momentum coefficient $\beta_1 = 0.9$, smoothing coefficient $\beta_2 = 0.95$, and $\varepsilon = 10^{-8}$. The learning rate (lr) schedule consists of a linear warmup followed by a constant phase and eventually linear cooldown spanning half of training (see examples in fig. 1). This schedule keeps the peak lr and decay phase identical for different warmup lengths. The learning rate and warmup length are optimized for various configurations. Our code builds on NanoGPT [14] with utilities from Kosson et al. [17], adopting NanoGPT's hyperparameters and base training setup.

Figure 1 shows the baseline performance for our setup. We observe that even short warmup can significantly improve performance. Not using warmup results in faster initial progress for a given learning rate, but eventually falls behind leaving a permanent gap. Warmup not only stabilizes higher learning rates, but also prevents a lasting degradation in the model performance.

3. Measuring & Controlling the Update Size

We will focus our analysis on the dot products making up neurons, e.g.:

$$\boldsymbol{y} = \boldsymbol{w}^\top \boldsymbol{X} = [y_1, \dots, y_B]^\top = [\langle \boldsymbol{w}, \boldsymbol{x}_1 \rangle, \dots, \langle \boldsymbol{w}, \boldsymbol{x}_B \rangle]^\top \tag{1}$$

where $\boldsymbol{y} \in \mathbb{R}^B$ is a batch of outputs, $\boldsymbol{X} \in \mathbb{R}^{C \times B}$ is a batch of inputs and $\boldsymbol{w} \in \mathbb{R}^C$ is the weight vector of the neuron. The weights are updated $\boldsymbol{w} \mapsto \boldsymbol{w} + \Delta\boldsymbol{w}$, which also causes an output change $\Delta\boldsymbol{y} = \Delta\boldsymbol{w}^\top \boldsymbol{X}$, computed on the same inputs. We can quantify the size of the update in weight space, i.e. in terms of $\Delta\boldsymbol{w}$, or in representation space with $\Delta\boldsymbol{y}$.

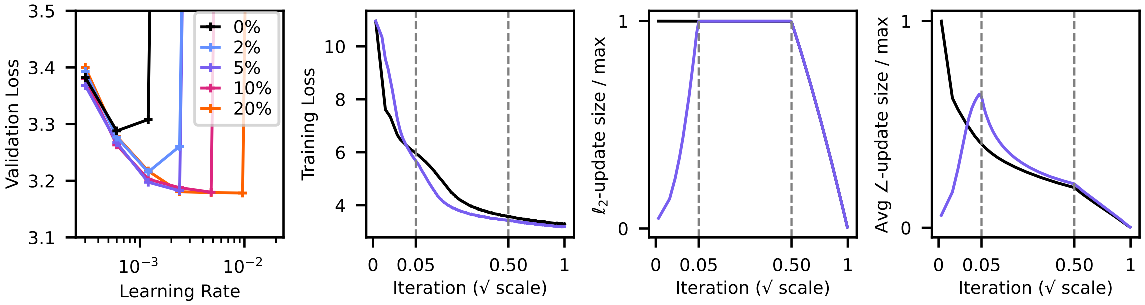

Figure 2: LionA (algo. 2) fails to significantly reduce the warmup advantage. **Panel 1:** Final validation loss across various learning rates and warmup percentages shows a reduced but still significant no-warmup penalty compared to AdamW (fig. 1). **Panel 2:** Training curves for 0% vs. 5% warmup at the highest stable learning rate for 0%, with warmup quickly overtaking no-warmup as before. **Panel 3:** LionA successfully controls the ℓ_2-update norm. **Panel 4:** Early angular updates (see §3) are large without warmup and do not follow the learning rate schedule throughout training.

The ℓ_2-update: $\|\boldsymbol{w}\|_2$ may be the simplest measure of the update size. In fig. 1 we see that the update magnitude of AdamW [22] varies significantly. We analyze this in appx. C, finding that the momentum bias-correction plays a large role. To control the update norm exactly, we can use normalization via the sign function as in Lion [3]. The hyperparameter configuration of Lion differs significantly from AdamW, so in appx. A we propose a variant LionA that is more compatible. Figure 2 shows that controlling the update size in this manner is not sufficient to eliminate warmup.

The angular update: $\angle(\boldsymbol{w}, \boldsymbol{w} + \Delta\boldsymbol{w}) = \arccos(\langle \boldsymbol{w}, \boldsymbol{w} + \Delta\boldsymbol{w}\rangle / (\|\boldsymbol{w}\|\|\boldsymbol{w} + \Delta\boldsymbol{w}\|))$ can be viewed as an "effective learning rate" accounting for the weight magnitude (see appx. D). We can approximately control the angular updates by fixing the weight magnitude via projections as proposed by Kosson et al. [17]. Combining this with LionA results in a rotational variant, LionAR (see appx. A, algo. 3 for exact formulation). This stabilizes training and significantly reduces the benefit of warmup as shown in fig. 3, but does not completely eliminate it without higher momentum (see later). Controlling the angular updates makes optimization invariant to certain aspects of the curvature (appx. D). High initial curvature is thought to be a major reason warmup is needed (appx. B).

The Relative Representation Change (RRC): $\|\Delta\boldsymbol{y}\|/\|\boldsymbol{y}\|$ is a similar measure as the angular update, but for the outputs of the neuron instead of its weights. We conjecture a large RRC may cause issues in the non-linearities, such as dead ReLUs or saturated sigmoids / softmax. Warmup could benefit training by preventing large RRC values as observed in fig. 3. When different samples $\boldsymbol{x}_b$ result in similar gradients $\boldsymbol{g}_b$ for $\boldsymbol{w}$, the RRC for a given angular update will be larger than otherwise. Defining the signal-to-noise ratio (SNR) of the gradient as $\varphi = \|\mathbb{E}[\boldsymbol{g}_b]\|^2/\text{trace}(\text{Cov}[\boldsymbol{g}_b])$, we show in appx. E that for normalized gradient descent with strong simplifying assumptions, we have:

$$\frac{\mathbb{E}[\|\Delta\boldsymbol{y}\|^2]}{\mathbb{E}[\|\boldsymbol{y}\|^2]} = \frac{\eta^2 C}{B^2\|\boldsymbol{w}\|^2}\frac{1}{\varphi+\frac{1}{B}}\left((\varphi+1) + \frac{B-1}{C} + \left(\frac{(B-1)^2\varphi}{\varphi+1}\left(\varphi+\frac{1}{C}\right) + 2(B-1)\varphi\right)\right) \quad (2)$$

If we treat the RRC as a trust region where the non-linearities are not strongly affected by a single update, this suggests that the learning rate should be downscaled for higher SNR values depending on the batch size. The first two panels of fig. 4 depict these scaling curves along with measurements of the SNR, showing high initial values. Scaling the update size of LionAR in this manner is sufficient

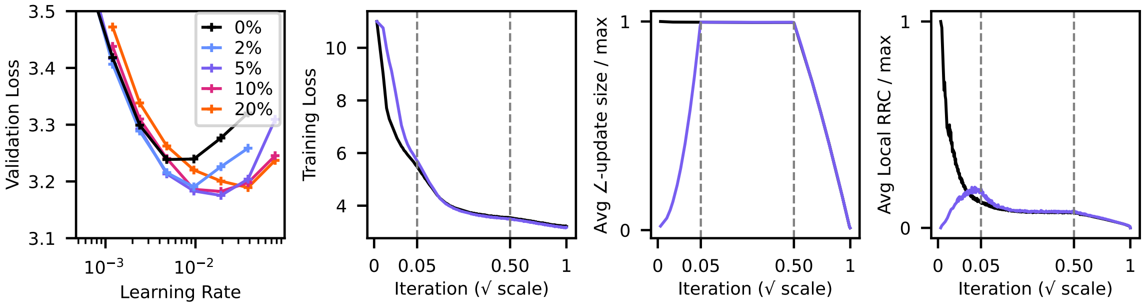

Figure 3: LionAR (algo. 3) reduces but does not fully eliminate the benefit of warmup. **Panel 1:** LionAR is more stable across learning rates and shows a reduced but still significant performance gap without warmup. **Panel 2:** Comparing the 0% and 5% warmup for learning rate $\approx 10^{-2}$ shows the warmup run overtaking early in training. **Panel 3:** LionAR precisely controls the angular update size throughout training. **Panel 4:** Despite fixed angular (and thus relative) updates in weight space, the relative change of the internal representations (see §3) is large initially without warmup.

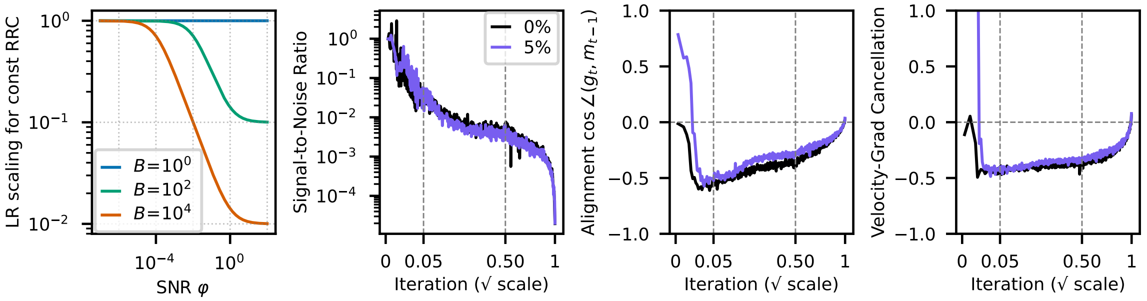

Figure 4: Equation (2) predicts that the learning rate needs to be downscaled for higher signal to noise ratios (φ) to keep the relative representation change constant (see appx. G.6). Larger batch sizes are affected more, with scaling becoming significant when $\varphi > B^{-1}$. **Panel 2:** Measurements of the SNR for the two highlighted runs in fig. 3. Note the SNR starts very high but is also remains large in comparison to our $B = 480$ for almost all of training. **Panel 3:** The gradient is strongly oppositely aligned with the momentum vector for most of training (shown for an example layer). **Panel 4:** Projecting the momentum component of the updates onto the gradient component shows that this results in the momentum vector "cancelling" roughly half the gradient on average.

to eliminate the benefit of warmup as shown in the first panel of fig. 5. This acts similar to an automatic warmup, but also distorts the learning rate schedule which can lead to issues, see appx. E. We believe directly controlling the RRC is promising but needs further development to be practical.

4. The Role of Momentum

Momentum is believed to be a key enabler of optimization with larger batch sizes [29, 30, 35]. Momentum spreads a gradient contribution out over multiple steps which tends to make each update smaller, especially for a random walk (see appx. G.1), which is reflected in the update scaling coefficients in our algorithms. The smaller updates are counteracted by an increased correlation in their direction, which can result in similar "long term" changes from each gradient sample, especially

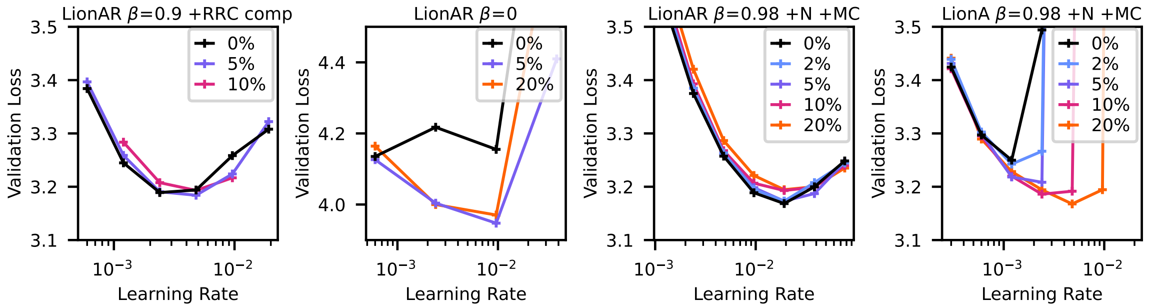

Figure 5: Panel 1: LionAR with a correction factor for the RRC based on eq. (2) does not benefit from a warmup. **Panel 2:** LionAR training without momentum results in drastically lower performance. **Panel 3:** In LionAR with higher momentum $\beta = 0.98$, Nesterov momentum and an inverse bias correction for early momentum, no warmup performs best. **Panel 4:** The same does not apply to LionA, suggesting these changes are insufficient without controlling the angular updates.

for simpler methods like SGD that don't normalize the step size. In the last two panels of fig. 4 we observe that in our setup the gradient and momentum are negatively correlated, counteracting each other. We find momentum crucial for performant training, panel 2 of fig. 5 shows significant degradation without it.

We believe the smaller update sizes for momentum combined with the potential for later gradients to counteract earlier gradients during their application over time, can help stabilize training. An otherwise large relative representation change is spread out over multiple steps and counteracted by later gradients. Higher values of momentum should amplify these effects. Looking at the total contribution of each gradient also implies that **with momentum early updates should be smaller when measured in parameter space, otherwise the relative representation change for those samples is too large.** This is equivalent to removing the β_1 bias correction in AdamW, or introducing *an inverse bias correction* in Lion like algorithms (see appx. G.1 for details). Higher β values should help amplify the stabilization effects of momentum. **In fig. 5 we find that at higher momentum values LionAR no longer benefits from warmup unlike LionA which still needs it**. These experiments use Nesterov momentum and the additional inverse bias correction, though these adjustments offer only minor improvements compared to higher momentum.

5. Conclusion

In this work, we explored why learning rate warmup benefits GPT training from the perspective of the update size. We demonstrated that measuring or controlling the update size in parameter space does generally not explain or replicate the advantages of using warmup. However, quantifying the update size in terms of the relative change in neural representations shows potential. This measure is closely linked to the angular update size but accounts for changes in the signal characteristics of the gradient, which can vary significantly throughout training. Effectively controlling neural representation changes is a challenging task we leave for future work, but our initial attempts show encouraging results in reducing the need for a manually configured warmup. We also highlighted the importance of high momentum for warmup; when combined with angular update control and an inverse bias correction, it may enable efficient warmup-free training. Overall, our work provides new insights into the necessity of learning rate warmup with modern optimizers beyond SGD and suggests potential directions for eliminating it in practice.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Jeremy Bernstein, Arash Vahdat, Yisong Yue, and Ming-Yu Liu. On the distance between two neural networks and the stability of learning. *Advances in Neural Information Processing Systems*, 33:21370–21381, 2020. arXiv:2002.03432.

[3] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V Le. Symbolic discovery of optimization algorithms. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=ne6zeqLFCZ`. arXiv:2302.06675.

[4] Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. Online normalization for training neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. arXiv:1905.05894.

[5] Jingwen Fu, Bohan Wang, Huishuai Zhang, Zhizheng Zhang, Wei Chen, and Nanning Zheng. When and why momentum accelerates sgd: An empirical study. *arXiv preprint arXiv:2306.09000*, 2023.

[6] Justin Gilmer, Behrooz Ghorbani, Ankush Garg, Sneha Kudugunta, Behnam Neyshabur, David Cardoze, George Edward Dahl, Zachary Nado, and Orhan Firat. A loss curvature perspective on training instabilities of deep learning models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=OcKMT-36vUs`. arXiv:2110.04369.

[7] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r14EOsCqKX`. arXiv:1810.13243.

[8] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. arXiv:1512.03385.

[10] 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. URL `https://arxiv.org/abs/2203.15556`.

[11] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*,

pages 4475–4483. PMLR, 2020. URL https://proceedings.mlr.press/v119/huang20f.html.

[12] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017. arXiv:1703.06868.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. arXiv:1502.03167.

[14] Andrej Karpathy. nanogpt. https://github.com/karpathy/nanoGPT/, 2023.

[15] Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and improving the training dynamics of diffusion models. *arXiv preprint arXiv:2312.02696*, 2023.

[16] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. arXiv:1412.6980.

[17] Atli Kosson, Bettina Messmer, and Martin Jaggi. Rotational Equilibrium: How weight decay balances learning across neural networks. *arXiv preprint arXiv:2305.17212*, 2023.

[18] Alex Krizhevsky. Learning multiple layers of features from tiny images. *self-published*, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[19] Zhiyuan Li, Kaifeng Lyu, and Sanjeev Arora. Reconciling modern deep learning with traditional optimization analyses: The intrinsic learning rate. *Advances in Neural Information Processing Systems*, 33:14544–14555, 2020. arXiv:2010.02916.

[20] Zhiyuan Li, Sadhika Malladi, and Sanjeev Arora. On the validity of modeling SGD with stochastic differential equations (SDEs). In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=goEdyJ_nVQI. arXiv:2102.12470.

[21] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgz2aEKDr. arXiv:1908.03265.

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7. arXiv:1711.05101.

[23] Kaifeng Lyu, Zhiyuan Li, and Sanjeev Arora. Understanding the generalization benefit of normalization layers: Sharpness reduction. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=xp5VOBxTxZ. arXiv:2206.07085.

[24] Jerry Ma and Denis Yarats. On the adequacy of untuned warmup for adaptive optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8828–8836, 2021. arXiv:1910.04209.

[25] Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. On the SDEs and scaling rules for adaptive gradient algorithms. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=F2mhzjHkQP`. arXiv:2205.10287.

[26] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[27] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *self-published*, 2019. URL `https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

[28] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016. arXiv:1602.07868.

[29] Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019. arXiv:1811.03600.

[30] Samuel Smith, Erich Elsen, and Soham De. On the generalization benefit of noise in stochastic gradient descent. In *International Conference on Machine Learning*, pages 9058–9067. PMLR, 2020. arXiv:2006.15081.

[31] Sebastian Stich, Amirkeivan Mohtashami, and Martin Jaggi. Critical parameters for scalable distributed learning with large batches and asynchronous updates. In *International Conference on Artificial Intelligence and Statistics*, pages 4042–4050. PMLR, 2021. arXiv:2103.02351.

[32] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013. URL `https://proceedings.mlr.press/v28/sutskever13.html`.

[33] 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. arXiv:1706.03762.

[34] Ruosi Wan, Zhanxing Zhu, Xiangyu Zhang, and Jian Sun. Spherical motion dynamics: Learning dynamics of normalized neural network using sgd and weight decay. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 6380–6391. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/326a8c055c0d04f5b06544665d8bb3ea-Paper.pdf`. arXiv:2006.08419.

[35] Runzhe Wang, Sadhika Malladi, Tianhao Wang, Kaifeng Lyu, and Zhiyuan Li. The marginal value of momentum for small learning rate SGD. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=3JjJezzVkT`. arXiv:2307.15196.

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

[37] Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, et al. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023. URL `https://arxiv.org/abs/2309.14322`.

[38] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. arXiv:1803.08494.

[39] 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. arXiv:2002.04745.

[40] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=Bx6qKuBM2AD`. arXiv:2203.03466.

[41] Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

[42] Dong Yin, Ashwin Pananjady, Max Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter Bartlett. Gradient diversity: a key ingredient for scalable distributed learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1998–2007. PMLR, 2018. arXiv:1706.05699.

[43] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[44] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Syx4wnEtvH`. arXiv:1904.00962.

[45] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? Insights from a noisy quadratic model. *Advances in Neural Information Processing Systems*, 32, 2019. arXiv:1907.04164.

Appendix A. Algorithms

The baseline AdamW is shown in algo. 1. A Lion-style analog is shown in algo. 2. A further rotational modification is made in algo. 3.

Algorithm 1 AdamW (PyTorch variant, differing from the original by Loshchilov and Hutter [22])

```
Require: Learning rate $\eta_t$, weight decay $\lambda$, momentum $\beta_1$, magnitude smoothing $\beta_2$, $\varepsilon$ for numerical stability
Initialize: Time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_0$, momentum vector $\boldsymbol{m}_0 \leftarrow 0$, magnitude vector $\boldsymbol{v}_0 \leftarrow 0$
while stopping criteria not met :
    $t \leftarrow t + 1$
    $\boldsymbol{g}_t \leftarrow$ Mini-batch gradient w.r.t. $\boldsymbol{\theta}_{t-1}$
    $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$
    $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
    $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$
    $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$
    $\boldsymbol{\theta}_t \leftarrow (1 - \eta_t \lambda)\boldsymbol{\theta}_{t-1} - \eta_t \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \varepsilon)$
```

Algorithm 2 LionA: A modified version of the Lion [3] optimizer for greater compatibility with AdamW (algo. 1). The sign operation replaces the magnitude smoothing, explicitly controlling the ℓ_2-norm of each update. Additional scaling keeps the hyperparameters comparable to AdamW.

```
Require: Learning rate $\eta_t$, weight decay $\lambda$, momentum $\beta$, Nesterov flag $\nu$
Initialize: Time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_0$, momentum vector $\boldsymbol{m}_0 \leftarrow 0$
while stopping criteria not met :
    $t \leftarrow t + 1$
    $\boldsymbol{g}_t \leftarrow$ Mini-batch gradient w.r.t. $\boldsymbol{\theta}_{t-1}$
    $\boldsymbol{m}_t \leftarrow \beta \boldsymbol{m}_{t-1} + (1 - \beta)\boldsymbol{g}_t$
    if Nesterov flag $\nu$ is set :
        $\boldsymbol{\theta}_t \leftarrow (1 - \eta_t \lambda)\boldsymbol{\theta}_{t-1} - \eta_t \cdot \sqrt{(1 - \beta^2)^2 + \beta^4 \frac{1-\beta}{1+\beta}} \cdot \mathrm{sign}(\beta \boldsymbol{m}_t + (1 - \beta)\boldsymbol{g}_t)$
    else:
        $\boldsymbol{\theta}_t \leftarrow (1 - \eta_t \lambda)\boldsymbol{\theta}_{t-1} - \eta_t \cdot \sqrt{\frac{1-\beta}{1+\beta}} \cdot \mathrm{sign}(\boldsymbol{m}_t)$
```

Appendix B. Related Work

The earliest use of learning rate warmup we are aware of was in ResNet [9], where a lower constant learning rate was applied at the start of training. Earlier works may have employed similar concepts; for example, Sutskever et al. [32] utilized a momentum schedule that could induce a similar effect in the "effective learning rate" as defined by Fu et al. [5]. The practice of linear warmup, in its current form, was popularized by Goyal et al. [8] and Vaswani et al. [33].

Warmup has been studied indirectly in various neural network optimizer works. A notable example is RAdam [21], a modification of Adam [16] aimed at reducing the need for warmup. However, Ma and Yarats [24] demonstrated that RAdam essentially incorporates a fixed warmup schedule within the optimizer. Relative optimizers like LARS [43] and LAMB [44] are also considered to reduce the necessity for warmup [19]. Bernstein et al. [2] propose a relative optimizer called Fromage and analyze how relative weight changes relate to relative representation changes, but differ from our approach in that they do not describe the effects of the gradient signal-to-noise ratio on this

Algorithm 3 LionAR: A rotational version of algo. 2 inspired by Kosson et al. [17]. The parameter vector is divided into sub-vectors $\boldsymbol{\theta} = [\boldsymbol{\theta}^{(1)}, \dots, \boldsymbol{\theta}^{(P)}]$, each corresponding to either the weight vector of a neuron (e.g. a matrix row / a convolutional filter), or other parameters such as gains and biases. The updates of neuronal weight vectors are scaled to be proportional to their magnitude which is kept constant through projections that replace weight decay. Additional hyperparameter adjustments are made for compatibility with AdamW. The weight decay hyperparameter remains, fulfilling its primary role as a scaling factor for the relative updates of neurons [17].

```
Require: Learning rate $\eta_t$, weight decay $\lambda$, momentum $\beta$, Nesterov flag $\nu$
Initialize: Time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_0$, momentum vector $\boldsymbol{m}_0 \leftarrow 0$
while stopping criteria not met :
    $t \leftarrow t + 1$
    $[\boldsymbol{g}_t^{(1)}, \dots, \boldsymbol{g}_t^{(P)}] \leftarrow$ Mini-batch gradient w.r.t. $\boldsymbol{\theta}_{t-1}$, divided into sub-vectors like $\boldsymbol{\theta}$
    for $p \in \{1, \dots, P\}$ :
        $\boldsymbol{m}_t^{(p)} \leftarrow \beta \boldsymbol{m}_{t-1}^{(p)} + (1-\beta)\boldsymbol{g}_t^{(p)}$
        if Nesterov flag $\nu$ is set :
            $\boldsymbol{u}_t^{(p)} \leftarrow \beta \boldsymbol{m}_t^{(p)} + (1-\beta)\boldsymbol{g}_t^{(p)}$
            $\gamma \leftarrow \sqrt{(1-\beta^2)^2 + \beta^4 \frac{1-\beta}{1+\beta}}$          # Nesterov momentum scaling factor
        else:
            $\boldsymbol{u}_t^{(p)} \leftarrow \boldsymbol{m}_t^{(p)}$
            $\gamma \leftarrow \sqrt{\frac{1-\beta}{1+\beta}}$          # Heavy-ball momentum scaling factor
        if $\boldsymbol{\theta}^{(p)} \in \mathbb{R}^C$ is a neuronal weight vector :
            $\hat{\boldsymbol{\theta}}_t^{(p)} \leftarrow \boldsymbol{\theta}_{t-1}^{(p)} - \frac{\eta_t}{\max_\tau(\eta_\tau)} \cdot \sqrt{2\max_\tau(\eta_\tau)\lambda} \cdot \gamma \cdot (\|\boldsymbol{\theta}_0^{(p)}\| / \sqrt{C}) \cdot \mathrm{sign}(\boldsymbol{u}_t^{(p)})$
            $\boldsymbol{\theta}_t^{(p)} \leftarrow \hat{\boldsymbol{\theta}}_t^{(p)} \cdot \|\boldsymbol{\theta}_0^{(p)}\| / \|\hat{\boldsymbol{\theta}}_t^{(p)}\|$          # Reset the magnitude to the initial value
        else:
            $\boldsymbol{\theta}_t^{(p)} \leftarrow \boldsymbol{\theta}_{t-1}^{(p)} - \eta_t \cdot \gamma \cdot \mathrm{sign}(\boldsymbol{u}_t^{(p)})$
```

relationship. We build upon the work of Kosson et al. [17] which showed that weight decay can make standard optimizers function as approximate relative optimizers and proposed optimizer variants that reduce the benefit of warmup without fully eliminating it.

The effect of warmup in transformers was empirically studied by Wortsman et al. [37]. Xiong et al. [39] proposed the pre-LN normalization placement for transformers, showing it reduces the need for warmup. Huang et al. [11] studied initialization in transformers showing a link to warmup.

Finally, warmup has been studied directly on its own. Gotmare et al. [7] studied the effect of warmup, finding it helps avoid overly large updates to the weights of later layers which could be frozen to achieve a similar benefit. Gilmer et al. [6] study the need for warmup from a curvature perspective, showing it may help "push" the optimization trajectory towards flatter regions where higher learning rates are stable. Smith et al. [30] arrive at a similar conclusion, there is a stable learning rate that varies throughout training based on the curvature which limits the learning rate early on, necessitating warmup. These works focus on SGD with momentum, but it is less clear how curvature affects Adam-like or relative optimizers (see discussion on angular updates).

The relation between stochastic gradient noise and learning rate has been studied in several works [20, 25, 26, 31, 42, 45]. They find that the update size can be increased roughly linearly with the batch size up to a certain *critical batch size* that depends on ratio of the mean and variance of the mini-batch gradient. We show how the signal-to-noise ratio (SNR) of the mini-batch gradient amplifies changes to the neural representations of a network given a normalized update in weight space. We observe that the SNR starts out high but decreases over time, which translates to large early changes in the internal representations without warmup.

Appendix C. The Interaction of Momentum and the ℓ_2-Update Norm in AdamW

Adam-like optimizers such as AdamW (algo. 1) differ from simpler methods like SGD with momentum in that they normalize the update size with the gradient magnitude. This makes them invariant to a rescaling of the loss function and helps counteract potential differences in the gradient magnitude between layers. Optimizers that do not have this property might diverge to infinity if a high learning rate is combined with large initial gradients or large curvature, as the update size is unbounded. Warmup can help stabilize SGD as previous works have shown [6, 8].

Although AdamW normalizes the update size based on the gradient, its magnitude can still vary throughout training as seen in fig. 1. This can be caused by changes in the gradient magnitude itself, especially when using different values of β_1 and β_2. However, it can also be caused by momentum and especially the bias correction (algo. 1, line 7). The magnitude of $\boldsymbol{m}_t$ depends on the alignment of subsequent gradients $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t$ whereas the normalization factor $\boldsymbol{v}_t$ does not. For example, when each $\boldsymbol{g}_t$ is an independent zero-mean random vector with a fixed second moment $\mathbb{E}[\boldsymbol{g}_t^2] = \boldsymbol{\sigma}^2$, we have (see appx. G.1 for details):

$$\mathbb{E}[\boldsymbol{m}_t^2] = (1 - \beta_1^{2t})\frac{1 - \beta_1}{1 + \beta_1}\boldsymbol{\sigma}^2, \qquad \mathbb{E}[\boldsymbol{v}_t] = (1 - \beta_2^t)\boldsymbol{\sigma}^2 \tag{3}$$

In this case the bias correction for β_1 is incorrect since it is derived for a constant gradient. With the bias correction the size becomes $\mathbb{E}[\|\hat{\boldsymbol{m}}\|^2] = \frac{1+\beta_1^t}{1-\beta_1^t}\frac{1-\beta_1}{1+\beta_1}\boldsymbol{\sigma}^2$, amplifying the norm of early updates by $\sqrt{(1 + \beta_1^t)/(1 - \beta_1^t)}$. This factor is larger if the gradients are negatively correlated, which we empirically observe often happens early in training.

AdamW does therefore not control the ℓ_2-norm of the update very well, due to the initial bias correction, changes in the alignment of the gradients throughout training and if the gradient norm is rapidly changing. Lion [3] is a closely related optimizer that uses an element-wise sign operation to normalize the update, giving $+1$ for positive values, -1 for negative values and 0 for zeros. Ignoring the possibility of zeros, this gives a constant update norm. Lion is closely related to Adam, and can be obtained by tracking the size of $\boldsymbol{m}_t$ instead of $\boldsymbol{g}_t$ in line 6 while setting $\beta_2 = 0$. It also uses Nesterov momentum instead of the traditional heavy-ball variant. Lion uses a slightly odd parameterization that differs significantly from AdamW, to keep the similarity we propose LionA (algo. 2). We scale the ℓ_2 update size to match that of AdamW in the random-gradient scenario, see appx. G.1 for the derivation of the scaling factors.

In fig. 2 we repeat the baseline sweep using LionA. **Despite perfect control of the ℓ_2 update norm (as seen in panel 3), the benefit of warmup remains. This leads us to conclude that the ℓ_2 update size is not sufficient to quantify the "effectively" large updates that we conjecture warmup mitigates.** The final panel shows that the angular update size (see definition in the following section), proposed to be a better measure of an effective step size by Wan et al. [34], still varies throughout training with a spike at the start of training.

Appendix D. The Importance and Irregularity of the Angular Update Size

The effect of a weight vector $\boldsymbol{w}_t \in \mathbb{R}^C$ used in a dot product with some vector $\boldsymbol{x}$ (e.g., in a neuron):

$$\langle \boldsymbol{w}_t, \boldsymbol{x} \rangle = \|\boldsymbol{w}_t\| \|\boldsymbol{x}\| \cos\left(\angle(\boldsymbol{w}_t, \boldsymbol{x})\right) \tag{4}$$

can be understood in terms of its magnitude $\|\boldsymbol{w}_t\|$ and direction $\boldsymbol{w}_t / \|\boldsymbol{w}_t\|$. The magnitude acts like a gain, scaling the outputs, whereas the direction determines which input representations $\boldsymbol{x}$ the system responds to. The angular update size [34] of an update $\boldsymbol{w}_t \mapsto \boldsymbol{w}_{t+1}$ is defined as

$$\angle(\boldsymbol{w}_{t+1}, \boldsymbol{w}_t) = \arccos\left(\frac{\langle \boldsymbol{w}_{t-1}, \boldsymbol{w}_{t+1} \rangle}{\|\boldsymbol{w}_t\| \|\boldsymbol{w}_t\|}\right) \tag{5}$$

and measures how fast the direction of $\boldsymbol{w}_t$ changes during training, and thus its "preference" for $\boldsymbol{x}$.

With BatchNorm [13] and similar operations [1, 4, 12, 38], a network can become invariant to the magnitude of weight vectors like $\boldsymbol{w}_t$, such that only the direction matters and the vector is said to be *scale-invariant*. WeightNorm [28] provides a good example of this, changing the system to:

$$\langle \boldsymbol{w}_t / \|\boldsymbol{w}_t\|, \boldsymbol{x} \rangle = \|\boldsymbol{x}\| \cos\left(\angle(\boldsymbol{w}_t, \boldsymbol{x})\right) \tag{6}$$

Note that although the system output is invariant to the magnitude $\|\boldsymbol{w}_t\|$, traditional optimizers are not. Scaling the value of a scale-invariant weight vector by a factor of $c > 0$, results in a gradient that is scaled by c^{-1} and curvature that is scaled by c^{-2} (see appx. G.2). For SGD this scales the angular update by c^{-2} and for Adam-like optimizers it is scaled by c^{-1}. With weight decay the magnitude of scale-invariant vectors trends towards a certain stable equilibrium value over time which also results in a specific expected angular update size as described by Kosson et al. [17], Wan et al. [34].

This has several important implications. Changing the initialization magnitude of scale-invariant weights will scale the angular updates over time for standard optimizers, resulting in effects similar to modifying the learning rate schedule. For small initial weight magnitudes compared to the equilibrium magnitude, the early angular updates will be large and these optimizers may benefit from learning rate warmup to counteract this. These effects also make the notion of "curvature" somewhat arbitrary as it can be scaled without changing the encoded function. Optimizers that specifically account for the weight magnitude would be invariant to these effects which may reduce the need for warmup from the traditional curvature perspective. Although standard transformers are not fully scale-invariant, the angular update insights still approximately hold for un-normalized weights [17].

In light of this, we modify LionA to better control the angular update size by making the updates to weight matrices proportional to their weight magnitude, resulting in algo. 3. We normalize the angular update size to match the equilibrium value, replacing weight decay with projections similar to Kosson et al. [17]. However, unlike their RVs, we make the angular updates proportional to the learning rate schedule which we found was necessary for good performance in our case. We also do not rely on additional exponential moving averages to control the angular update size, instead utilizing the fixed update size from the LionA optimizer. This is similar to the Adam scheme used by Karras et al. [15] with good results for diffusion models. No additional normalization operations or scaling factors are introduced, which we still find to result in decent performance.

Figure 3 repeats the GPT2 training sweep with LionAR. Consistent with the findings of Kosson et al. [17] **we find that controlling the angular updates stabilizes training and decreases the benefit from warmup, but does not eliminate it in this setting**. Both the angular change and the

ℓ_2-norm are simple measures of the update magnitude in parameter space that do not account for the direction or other aspects of the update. In the next section we show how a fixed update size in parameter space can result in large changes to the internal representations of the network (a.k.a. features, activations etc), as shown in the final panel of fig. 3.

Appendix E. Early Gradient Alignment Results in Large Representation Changes

Measuring and controlling the update size in weight space failed to explain the need for warmup. As an alternative to the parameters, we can analyze changes in the internal representations or activations of the neural network. Although this is harder to analyze and control, it may ultimately be a better measure of the true impact of an update. A parameter update can only affect the network output, and hence the loss, by changing the representation of the network inputs at some layer. Large changes in the representations could significantly affect the non-linearities, potentially causing lasting issues such as dead ReLUs or vanishing gradients from saturated sigmoids. This could in turn explain the lasting performance degradation observed without warmup.

A given parameter update will affect the representations of each distinct input sample differently. The gradients computed on these samples also generally differ, but can align to some extent. For a higher gradient alignment, the impact of a parameter update of a given magnitude on the representations will be larger than otherwise. We will analyze this for the dot product of a neuron:

$$\boldsymbol{y} = \boldsymbol{w}^\top \boldsymbol{X} = [y_1, \dots, y_B]^\top = [\langle \boldsymbol{w}, \boldsymbol{x}_1 \rangle, \dots, \langle \boldsymbol{w}, \boldsymbol{x}_B \rangle]^\top \tag{7}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_B] \in \mathbb{R}^{C \times B}$ are the C-dimensional representations of a random mini-batch of B inputs that is fed into the neuron, $\boldsymbol{w} \in \mathbb{R}^C$ is the weight vector, and $\boldsymbol{y} \in \mathbb{R}^B$ is a batch of outputs. For a weight update $\boldsymbol{w} \mapsto \boldsymbol{w} + \Delta\boldsymbol{w}$, we aim to quantify the size of the output change $\Delta\boldsymbol{y} = \Delta\boldsymbol{w}^\top \boldsymbol{X}$ computed on the same inputs. We focus on the *Relative Representation Change (RRC)*:

$$\frac{\|\Delta \boldsymbol{y}\|}{\|\boldsymbol{y}\|} = \frac{\|\Delta \boldsymbol{w}^\top \boldsymbol{X}\|}{\|\boldsymbol{w}^\top \boldsymbol{X}\|} \tag{8}$$

similar to the angular weight updates, as the sensitivity to the absolute change $\|\Delta\boldsymbol{y}\|$ can be unclear due to normalization or other scaling operations. Note that this is a measure of a *local change*, not accounting for changes in the inputs $\boldsymbol{X}$ from updates to preceding layers (*global change*).

Our analysis focuses on the relatively tractable case of normalized gradient descent with updates:

$$\Delta \boldsymbol{w} = -\eta \frac{\boldsymbol{g}}{\sqrt{\mathbb{E}[\|\boldsymbol{g}\|^2]}}, \qquad \boldsymbol{g} = \frac{1}{B} \sum_{b=1}^{B} \boldsymbol{g}_b \tag{9}$$

where $\boldsymbol{g}_b$ is the gradient of some loss w.r.t. $\boldsymbol{w}$ for the b-th element of the mini-batch. We will use the following definitions, properties, lemmas and assumptions for this system (see appx. G.4 for details):

- D1: We define $\boldsymbol{g}_b =: \bar{\boldsymbol{g}} + \tilde{\boldsymbol{g}}_b$ where $\bar{\boldsymbol{g}} = \mathbb{E}[\boldsymbol{g}]$ and $\tilde{\boldsymbol{g}}_b$ is the difference with $\mathbb{E}[\tilde{\boldsymbol{g}}_b] = \boldsymbol{0}$.
- D2: We define $\varphi := \mathbb{E}[\|\bar{\boldsymbol{g}}\|^2] / \mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2]$ as the Signal-to-Noise Ratio (SNR) of the gradient.
- P1: For a neuron, $\boldsymbol{g}_b \parallel \boldsymbol{x}_b$, and hence $\boldsymbol{x}_b = \operatorname{sign}(\langle \boldsymbol{x}_b, \boldsymbol{g}_b \rangle) \cdot (\|\boldsymbol{x}_b\| / \|\boldsymbol{g}_b\|) \cdot (\bar{\boldsymbol{g}} + \tilde{\boldsymbol{g}}_b)$.

- L1: Consider two independent random vectors $\boldsymbol{a} \in \mathbb{R}^C$ and $\boldsymbol{b} \in \mathbb{R}^C$, whose elements are independent and identically distributed (IID). If at least one of the vectors has a zero-mean distribution, then the expected value of the squared inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$ is given by $\mathbb{E}[\langle \boldsymbol{a}, \boldsymbol{b}\rangle^2] = \mathbb{E}[\|\boldsymbol{a}\|^2]\mathbb{E}[\|\boldsymbol{b}\|^2]/C$.
- A1: We assume the following vector pairs satisfy L1: $(\boldsymbol{x}_i, \tilde{\boldsymbol{g}}_b)$ when $i \neq b$, $(\bar{\boldsymbol{g}}, \tilde{\boldsymbol{g}}_b)$ and $(\boldsymbol{w}, \boldsymbol{x}_b)$.

This allows us to compute the expected square relative representation change (appx. G.4 for details):

$$\begin{aligned}\frac{\mathbb{E}[(\Delta y_b)^2]}{\mathbb{E}[y_b^2]} &= \frac{\eta^2 C}{B^2\|\boldsymbol{w}\|^2}\frac{1}{\mathbb{E}[\|\boldsymbol{g}\|^2]}\Bigg(\mathbb{E}[\|\boldsymbol{g}_b\|^2] + \frac{B-1}{C}\mathbb{E}[\|\tilde{\boldsymbol{g}}_i\|^2] \\ &\qquad + \frac{(B-1)^2}{\mathbb{E}[\|\boldsymbol{g}_b\|^2]}\left(\|\bar{\boldsymbol{g}}\|^4 + \frac{\|\bar{\boldsymbol{g}}\|^2\mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2]}{C}\right) + 2(B-1)\|\bar{\boldsymbol{g}}\|^2\Bigg) \qquad (10) \\ &= \frac{\eta^2 C}{B^2\|\boldsymbol{w}\|^2}\frac{1}{\varphi + \frac{1}{B}}\left((\varphi+1) + \frac{B-1}{C} + \left(\frac{(B-1)^2\varphi}{\varphi+1}\left(\varphi + \frac{1}{C}\right) + 2(B-1)\varphi\right)\right) \qquad (11)\end{aligned}$$

The expected relative change in the output for a given sample can be broken down into three sources, the contribution of the sample itself (first term), random interference from the "noise" $\tilde{\boldsymbol{g}}_i$ of other samples (second term), and finally amplification of the common mean component $\bar{\boldsymbol{g}}$ (third term).

The RRC expression provides many interesting insights. In the case of large input dimension $C \to \infty$ and small SNR $\varphi \approx 0$, keeping the RRC constant for different batch sizes involves scaling the learning rate $\eta \propto \sqrt{B}$, as suggested by Malladi et al. [25] for Adam. When the SNR φ is some finite and value and C is still large, this scaling rule instead starts to break down around $B = 1/\varphi$, matching the predicted critical batch size of e.g. McCandlish et al. [26]. The role of the dimension C in the expression is curious, suggesting that narrower layers experience larger changes due to random inference from other samples in a given batch. The C in the leading factor also suggests that the angular updates can be smaller for a larger input dimension, similar to what is proposed in μ-parameterization [40, 41]. Most importantly, **this expression shows that if the SNR changes throughout training the learning rate needs to be adjusted to keep the RRC constant. In particular, with large batch sizes, a high initial SNR results in large representation changes which warmup can help prevent.** The first panel of fig. 4 shows how eq. (11) predicts we should downscale the learning rate for different batch sizes and SNRs, assuming we originally scaled the learning rate $\eta \propto \sqrt{B}$ and that C is large. The second panel confirms that the SNR indeed starts out large, suggesting lower initial learning rates are needed, i.e. warmup.

In the first panel of fig. 5, we show the results of adding a term that scales the update size as predicted by eq. (11). **This acts similar to an automatic warmup based on online measurements of the SNR, which we obtain from the gradient accumulation of micro-batches (appx. G.5).** Although this helps close the gap between warmup and no-warmup, the overall performance is slightly worse. One potential issue is that our batch size of 480 is quite large compared to the measured SNR, exceeding the critical batch size estimation throughout most of training. This results in a scaling of the step size throughout training, which distorts the decay phase. It also requires large learning rate values to counteract the scaling, which may destabilize the training of non-matrix weights like gains. We increase the weight decay by a factor of 32× to try to increase the angular updates relative to gains in order to compensate, but this value was not tuned and is unlikely to be

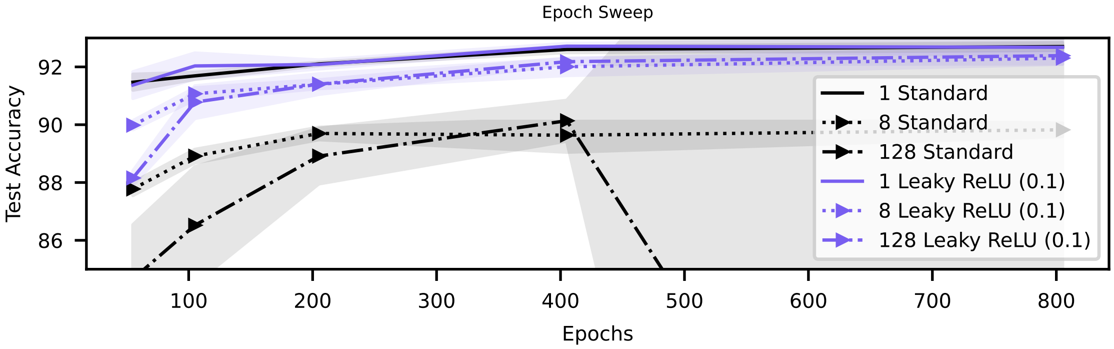

Figure 6: The performance gap due to large updates at the start of training cannot be closed with longer training for a standard ResNet-20. We suspect this is related to the non-linearities of the network. The experiment indicates that training with Leaky ReLU exhibits a smaller performance degradation from large initial updates.

optimal. We believe directly controlling the RRC is a promising direction but requires further work to be practical.

Appendix F. The Detrimental Effects of Large Updates

To investigate the effects of large updates at the beginning of training, we conducted controlled experiments on a ResNet-20 model on CIFAR-10 [18] due to resource constraints. We controlled the expected angular update throughout training using the rotational optimizer variant of AdamW proposed by Kosson et al. [17]. For the first 5 epochs, we applied either a standard learning rate of 0.05 or a notably increased learning rate by a factor of 8 or 128. For all experiments, we used a weight decay of 0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$, 5 warmup epochs, and trained for 205 epochs unless specified otherwise. The data was pre-processed by normalizing it with a mean of $(0.4914, 0.4822, 0.4465)$ and a standard deviation of $(0.2023, 0.1994, 0.2010)$ and applying simple data augmentation techniques as described by He et al. [9]. To run the experiment, we used the codebase from Wightman [36] and extended the utilities from Kosson et al. [17].

As shown in fig. 6, the performance of standard training does not recover when large updates are used at the beginning of training, even when the training time is extended to four times the normal duration for ReLU networks. This effect is less pronounced when replacing ReLUs with leaky ReLUs, suggesting that the non-linearities in the network might substantially impact the observed performance degradation.

The observation that larger initial updates result in more dead ReLUs later in training, as seen in the left figure of fig. 7, supports this hypothesis. This effect can be mitigated by freezing the biases at the beginning of training, as shown in the table in fig. 7.

Interestingly, we did not find a connection to overfitting to a small number of samples at the beginning of training. The performance of 92.1 can be recovered in this case. Additionally, we explored stable rank measurements as a potential factor but did not find a notable connection, as detailed in fig. 8.

Method	Speed 1	Speed 8	Speed 128
Standard (S)	92.13±0.2	89.48±0.2	88.22±1.6
Standard frozen bias (S-fb)	92.30±0.3	92.08±0.3	92.30±0.2
Random (R)	92.05±0.3	91.74±0.2	89.54±0.3
Random frozen bias (R-fb)	92.27±0.2	92.12±0.4	92.20±0.1
Leaky Relu 0.1	92.16±0.3	91.48±0.3	91.82±0.4
Leaky ReLU 0.1 frozen bias	92.46±0.2	92.49±0.1	92.35±0.2

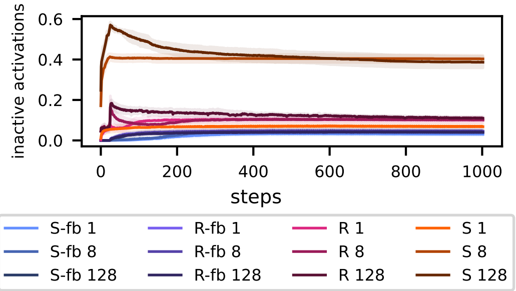

Figure 7: Comparison of different methods' performance across varying large updates and ratio of dead ReLUs. We observe a notable correspondence between larger ratio of dead ReLUs in the ResNet-20 and performance degradation as seen in the Table on the left.

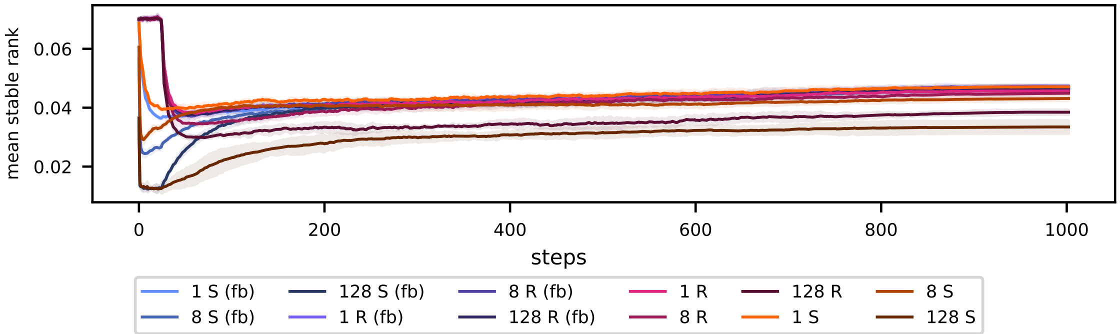

Figure 8: Impact on stable rank of varying large updates at the beginning of training on a standard ResNet-20. The stable rank seems to vary only minimally across different setups, except when using extremly large updates (increasing the learning rate by a factor of 128) and biases are not frozen.

Appendix G. Additional Mathematical and Technical Details

G.1. The magnitude of the Momentum Vector

Let's assume a scalar gradient g_t (e.g. for some coordinate) that is a random variable that is independent across time and has a zero mean distribution that does not change across time, i.e. $E[g_t] = 0$ and $\mathbb{E}[g_t^2] = \sigma^2$. For standard **heavyball-momentum** m_t, with $m_0 = 0$ and coefficient β (equivalent to β_1 for Adam) we have:

$$\mathbb{E}[m_t^2] = \mathbb{E}[(\beta m_{t-1} + (1-\beta) g_t)^2] \tag{12}$$

$$= \mathbb{E}\left[\left((1-\beta)\sum_{i=0}^{t-1} \beta^i g_{t-i}\right)^2\right] \tag{13}$$

$$= \mathbb{E}\left[(1-\beta)^2 \sum_{i=0}^{t-1} \beta^{2i} g_{t-i}^2 + (1-\beta)^2 \sum_{j=0}^{t-1} \sum_{\substack{k=0 \\ k\neq j}}^{t-1} \beta^{2t-j-k} g_{t-j} g_{g-k}\right] \tag{14}$$

$$= (1-\beta)^2 \sum_{i=0}^{t-1} \beta^{2i} \mathbb{E}[g_{t-i}^2] + (1-\beta)^2 \sum_{j=0}^{t-1} \sum_{\substack{k=0 \\ k \neq j}}^{t-1} \beta^{2t-j-k} \mathbb{E}[g_{t-j}]\mathbb{E}[g_{g-k}] \tag{15}$$

$$= (1-\beta)^2 \sum_{i=0}^{t-1} \beta^{2i} \sigma^2 + 0 \tag{16}$$

$$= (1-\beta)^2 \frac{1-\beta^{2t}}{1-\beta^2} \sigma^2 \tag{17}$$

$$= (1-\beta)^2 \frac{1-\beta^{2t}}{(1-\beta)(1+\beta)} \sigma^2 \tag{18}$$

$$= (1-\beta^{2t}) \frac{1-\beta}{1+\beta} \sigma^2 \tag{19}$$

In the limit $t \to \infty$ we have $(1-\beta^{2t}) \to 1$. We can derive the size of the second-moment v_t in AdamW in an analogous way, obtaining $\mathbb{E}[v_t] = (1-\beta_2^t)\sigma^2$. For a random walk, the update size of Adam is scaled in a similar way. Since the update size of Lion is fixed and does not depend on β, we scale the update size to match that of AdamW for a random walk in a steady state, i.e. by $\gamma = \sqrt{\frac{1-\beta}{1+\beta}}$ as seen in algo. 2.

Nesterov momentum: The update is modified to use

$$u_t = \beta m_t + (1-\beta) g_t \tag{20}$$

$$= \beta\left(\beta m_{t-1} + (1-\beta) g_t\right) + (1-\beta) g_t \tag{21}$$

$$= \beta^2 m_{t-1} + (1-\beta)(1+\beta) g_t \tag{22}$$

Note that m_{t-1} and g_t are independent and zero-mean, allowing us to use the previous result for:

$$\mathbb{E}[u_t^2] = \mathbb{E}\left[\left(\beta^2 m_{t-1} + (1-\beta)(1+\beta) g_t\right)^2\right] \tag{23}$$

$$= \beta^4 \mathbb{E}[m_{t-1}^2] + (1-\beta^2)^2 \mathbb{E}[g_t^2] \tag{24}$$

$$= \beta^4 (1-\beta^{2t-2}) \frac{1-\beta}{1+\beta} \sigma^2 + (1-\beta^2)^2 \sigma^2 \tag{25}$$

In the limit $t \to \infty$ this gives the Nesterov scaling factor used in LionA (algo. 2) to ensure that the update size corresponds to that of AdamW using an analogous Nesterov update.

Inverse bias correction for momentum: Adam uses a bias correction to attempt to fix the update size over time. This scales early updates resulting in the contributions of the corresponding gradients being amplified. The relative representation change for those samples is increased as a result, similar to applying the same update multiple times. Removing the β_1 bias correction from AdamW removes this effect. LionA and LionAR similarly scale the update size, making it constant. We can counteract this by changing our scaling factors to use the time varying expressions based on the derivations above. Note however, that this assumed the gradients were uncorrelated so it only approximately undoes the scaling effect for real values with arbitrary alignment of successive gradients. To summarize, the inverse bias correction for momentum changes the momentum scaling factors (γ in algo. 3) to vary over time:

$$\text{Nesterov:} \qquad \gamma_t = \sqrt{(1-\beta^2)^2 + (1-\beta^{2t-2})\beta^4 \frac{1-\beta}{1+\beta}} \tag{26}$$

$$\text{Heavy-ball:} \qquad \gamma_t = \sqrt{(1-\beta^{2t})\frac{1-\beta}{1+\beta}} \tag{27}$$

G.2. Properties of Scale Invariance

Derivations for the gradient magnitude and curvature can be found in existing works, for example Lyu et al. [23]. When a scale invariant weight is scaled by a factor $c > 0$, the gradient is scaled by c^{-1} which scales the ratio of the gradient norm and weight norm, and therefore the angular updates, by c^{-2}. For normalized optimizers like Adam and Lion, where the update norm is not affected by the gradient magnitude, this factor is decreased to c^{-1}.

G.3. The Angular Update Size in LionAR

The scaling factor for the angular update size in algo. 3 is adopted directly from the AdamW value derived by Kosson et al. [17]. Since the Nesterov momentum does not change the total contribution of each gradient it does not affect the equilibrium magnitude. The expected angular updates are therefore scaled in the same way as the RMS update norm we derived in appx. G.1.

G.4. Relative Representation Change for Normalized Gradient Descent

Property (P1): For a dot product $y = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$ and loss $\mathscr{L}(\boldsymbol{x}_b)$ that depends on y, we have:

$$\frac{\partial \mathscr{L}(\boldsymbol{x}_b)}{\partial \boldsymbol{w}} = \frac{\partial \mathscr{L}(\boldsymbol{x}_b)}{\partial y}\frac{\partial y}{\partial \boldsymbol{w}} = \frac{\partial \mathscr{L}(\boldsymbol{x}_b)}{\partial y}\boldsymbol{x}_b \tag{28}$$

where $\frac{\partial \mathscr{L}(\boldsymbol{x}_b)}{\partial y}$ is a scalar, ensuring that $\boldsymbol{g}_b := \frac{\partial \mathscr{L}(\boldsymbol{x}_b)}{\partial \boldsymbol{w}} \parallel \boldsymbol{x}_b$, assuming the vectors are not zero.

Lemma (L1): Consider two independent random vectors $\boldsymbol{a} \in \mathbb{R}^C$ and $\boldsymbol{b} \in \mathbb{R}^C$, whose elements are independent and identically distributed (IID). If at least one of the vectors has a zero-mean distribution, then the expected value of the squared inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$ is given by:

$$\mathbb{E}[\langle \boldsymbol{a}, \boldsymbol{b} \rangle^2] = \frac{\mathbb{E}[\|\boldsymbol{a}\|^2]\mathbb{E}[\|\boldsymbol{b}\|^2]}{C} \tag{29}$$

Proof: Let $\boldsymbol{a} = (a_1, a_2, \ldots, a_C)$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_C)$. The inner product $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ is given by:

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \sum_{i=1}^{C} a_i b_i.$$

We need to find $\mathbb{E}[\langle \boldsymbol{a}, \boldsymbol{b} \rangle^2]$. Expanding the square of the inner product:

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle^2 = \left(\sum_{i=1}^{C} a_i b_i\right)^2 = \sum_{i=1}^{C}\sum_{j=1}^{C} a_i b_i a_j b_j.$$

Taking the expectation, we get:

$$\mathbb{E}[\langle \boldsymbol{a}, \boldsymbol{b} \rangle^2] = \mathbb{E}\left[\sum_{i=1}^{C} \sum_{j=1}^{C} a_i b_i a_j b_j\right] = \sum_{i=1}^{C} \sum_{j=1}^{C} \mathbb{E}[a_i b_i a_j b_j].$$

Since $\boldsymbol{a}$ and $\boldsymbol{b}$ are independent and their elements are IID, we have:

$$\mathbb{E}[a_i b_i a_j b_j] = \mathbb{E}[a_i a_j]\mathbb{E}[b_i b_j].$$

Consider two cases:
1. When $i = j$:

$$\mathbb{E}[a_i b_i a_i b_i] = \mathbb{E}[a_i^2]\mathbb{E}[b_i^2].$$

2. When $i \neq j$:

$$\mathbb{E}[a_i b_i a_j b_j] = \mathbb{E}[a_i]\mathbb{E}[b_i]\mathbb{E}[a_j]\mathbb{E}[b_j].$$

Given that at least one of $\boldsymbol{a}$ or $\boldsymbol{b}$ has a zero-mean distribution, say $\boldsymbol{a}$ without loss of generality, we have $\mathbb{E}[a_i] = 0$. Thus:

$$\mathbb{E}[a_i b_i a_j b_j] = 0.$$

So, the expectation simplifies to:

$$\mathbb{E}[\langle \boldsymbol{a}, \boldsymbol{b} \rangle^2] = \sum_{i=1}^{C} \mathbb{E}[a_i^2]\mathbb{E}[b_i^2].$$

Since a_i and b_i are IID, we have:

$$\mathbb{E}[a_i^2] = \mathbb{E}[a_1^2] \quad \text{and} \quad \mathbb{E}[b_i^2] = \mathbb{E}[b_1^2].$$

Therefore:

$$\mathbb{E}[\langle \boldsymbol{a}, \boldsymbol{b} \rangle^2] = C\mathbb{E}[a_1^2]\mathbb{E}[b_1^2].$$

Recognizing that:

$$\mathbb{E}[\|\boldsymbol{a}\|^2] = \mathbb{E}\left[\sum_{i=1}^{C} a_i^2\right] = C\mathbb{E}[a_1^2],$$

$$\mathbb{E}[\|\boldsymbol{b}\|^2] = \mathbb{E}\left[\sum_{i=1}^{C} b_i^2\right] = C\mathbb{E}[b_1^2],$$

we have:

$$\mathbb{E}[a_1^2] = \frac{\mathbb{E}[\|\boldsymbol{a}\|^2]}{C} \quad \text{and} \quad \mathbb{E}[b_1^2] = \frac{\mathbb{E}[\|\boldsymbol{b}\|^2]}{C}.$$

Thus:

$$\mathbb{E}[\langle \boldsymbol{a}, \boldsymbol{b} \rangle^2] = C\left(\frac{\mathbb{E}[\|\boldsymbol{a}\|^2]}{C}\right)\left(\frac{\mathbb{E}[\|\boldsymbol{b}\|^2]}{C}\right) = \frac{\mathbb{E}[\|\boldsymbol{a}\|^2]\mathbb{E}[\|\boldsymbol{b}\|^2]}{C}.$$

This completes the proof.

Assumption (A1): We assume the following vector pairs satisfy L1: $(\boldsymbol{x}_i, \tilde{\boldsymbol{g}}_b)$ when $i \neq b$, $(\bar{\boldsymbol{g}}, \tilde{\boldsymbol{g}}_b)$ and $(\boldsymbol{w}, \boldsymbol{x}_b)$.

Vector pairs of the type $(\boldsymbol{x}_i, \tilde{\boldsymbol{g}}_b)$ and $(\bar{\boldsymbol{g}}, \tilde{\boldsymbol{g}}_b)$ should be independent and $\tilde{\boldsymbol{g}}_b$ has a zero mean distribution. However, the elements of each vector are not necessarily IID. For $(\boldsymbol{w}, \boldsymbol{x}_b)$, this is an even stronger assumption. Generally, neither $\boldsymbol{w}$ nor $\boldsymbol{x}_b$ is guaranteed to be IID or zero mean, and their independence later in training does not necessarily hold. Applying weight standardization to $\boldsymbol{w}$ or batch normalization to $\boldsymbol{x}$ would suffice to make this hold. Overall, this assumption can be viewed as a simplifying approximation to obtain reasonable predictions without additional information about these vectors.

Deriving the Relative Representation Change: Applying L1 directly gives us the original expected square output :

$$\mathbb{E}[y_b^2] = \mathbb{E}[\langle \boldsymbol{w}, \boldsymbol{x}_b \rangle^2] = \frac{\|\boldsymbol{w}\|^2 \mathbb{E}[\|\boldsymbol{x}_b\|^2]}{C} \tag{30}$$

For the expected square representation change we get:

$$\mathbb{E}[(\Delta y_b)^2] \tag{31}$$

$$= \mathbb{E}[\langle -\eta \boldsymbol{g} / \sqrt{\mathbb{E}[\|\boldsymbol{g}\|^2]}, \boldsymbol{x}_b \rangle^2] \tag{32}$$

$$= \frac{\eta^2}{B^2} \frac{1}{\mathbb{E}[\|\boldsymbol{g}\|^2]} \mathbb{E}\left[\left(\sum_{i=1}^{B} \langle \boldsymbol{g}_i, \boldsymbol{x}_b \rangle \right)^2 \right] \tag{33}$$

$$= \frac{\eta^2}{B^2} \frac{1}{\mathbb{E}[\|\boldsymbol{g}\|^2]} \mathbb{E}\left[\left(\mathrm{sign}(\langle \boldsymbol{x}_b, \boldsymbol{g}_b \rangle) \|\boldsymbol{g}_b\| \|\boldsymbol{x}_b\| + \sum_{i \neq B} \langle \boldsymbol{g}_i, \boldsymbol{x}_b \rangle \right)^2 \right] \tag{34}$$

$$= \frac{\eta^2}{B^2} \frac{1}{\mathbb{E}[\|\boldsymbol{g}\|^2]} \mathbb{E}\left[\left(\mathrm{sign}(\langle \boldsymbol{x}_b, \boldsymbol{g}_b \rangle) \|\boldsymbol{g}_b\| \|\boldsymbol{x}_b\| + (B-1) \langle \bar{\boldsymbol{g}}, \boldsymbol{x}_b \rangle + \sum_{i \neq b} \langle \tilde{\boldsymbol{g}}_i, \boldsymbol{x}_b \rangle \right)^2 \right] \tag{35}$$

$$\tag{36}$$

where we have used the definitions from eq. (9) and D1. Using property P1, we can write:

$$\langle \bar{\boldsymbol{g}}, \boldsymbol{x}_b \rangle = \left\langle \bar{\boldsymbol{g}}, \quad \mathrm{sign}(\langle \boldsymbol{x}_b, \boldsymbol{g}_b \rangle) \frac{\|\boldsymbol{x}_b\|}{\|\boldsymbol{g}_b\|} \cdot (\bar{\boldsymbol{g}} + \tilde{\boldsymbol{g}}_b) \right\rangle \tag{37}$$

$$= \mathrm{sign}(\langle \boldsymbol{x}_b, \boldsymbol{g}_b \rangle) \frac{\|\boldsymbol{x}_b\|}{\|\boldsymbol{g}_b\|} (\|\bar{\boldsymbol{g}}\|^2 + \langle \bar{\boldsymbol{g}}, \tilde{\boldsymbol{g}}_b \rangle) \tag{38}$$

Plugging this into the previous expression yields $\mathbb{E}[(\Delta y_b)^2]$

$$= \frac{\eta^2}{B^2} \frac{1}{\mathbb{E}[\|\boldsymbol{g}\|^2]} \mathbb{E}\left[\left(\mathrm{sign}(\langle \boldsymbol{x}_b, \boldsymbol{g}_b \rangle) \Big(\|\boldsymbol{g}_b\| \|\boldsymbol{x}_b\| + (B-1) \frac{\|\boldsymbol{x}_b\|}{\|\boldsymbol{g}_b\|} (\|\bar{\boldsymbol{g}}\|^2 + \langle \bar{\boldsymbol{g}}, \tilde{\boldsymbol{g}}_b \rangle) \Big) + \sum_{i \neq b} \langle \tilde{\boldsymbol{g}}_i, \boldsymbol{x}_b \rangle \right)^2 \right] \tag{39}$$

Squaring the expression results in various cross but all remaining dot products except the sign one are zero in expectation (due to the noise $\tilde{\boldsymbol{g}}$) and independent from each other. The cross terms involving these thus all disappear under the expectation. We apply Lemma L1 to their squares and approximate the expected norms of $\boldsymbol{x}_b$ and $\boldsymbol{g}_b$ as being independent. This gives $\mathbb{E}[(\Delta y_b)^2]$

$$\begin{aligned} &= \frac{\eta^2}{B^2}\frac{\mathbb{E}[\|\boldsymbol{x}_b\|^2]}{\mathbb{E}[\|\boldsymbol{g}\|^2]}\left(\mathbb{E}[\|\boldsymbol{g}_b\|^2] + \frac{(B-1)^2\|\bar{\boldsymbol{g}}\|^2}{\mathbb{E}[\|\boldsymbol{g}\|^2]}\left(\|\bar{\boldsymbol{g}}\|^2 + \frac{\mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2]}{C}\right)\right. \qquad (40)\\ &\qquad \left. +2(B-1)\|\bar{\boldsymbol{g}}\|^2 + \frac{B-1}{C}\mathbb{E}[\|\tilde{\boldsymbol{g}}_i\|^2]\right) \qquad (41)\end{aligned}$$

We can compute the expected magnitude of the batch gradient as:

$$\mathbb{E}[\|\boldsymbol{g}\|^2] = \mathbb{E}[\|\frac{1}{B}\sum_{i=1}^{B}(\bar{\boldsymbol{g}} + \tilde{\boldsymbol{g}}_i)\|^2] = \mathbb{E}[\|(\bar{\boldsymbol{g}} + \frac{1}{B}\sum_{i=1}^{B}\tilde{\boldsymbol{g}}_i)\|^2] = \|\bar{\boldsymbol{g}}\|^2 + \frac{1}{B}\mathbb{E}[\|\boldsymbol{g}_i\|^2] \tag{42}$$

and similarly $\mathbb{E}[\|\boldsymbol{g}_b\|^2] = \|\bar{\boldsymbol{g}}\|^2 + \mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2]$. Using these facts we can further write $\mathbb{E}[(\Delta y_b)^2]$

$$\begin{aligned} &= \frac{\eta^2}{B^2}\frac{\mathbb{E}[\|\boldsymbol{x}_b\|^2]}{\mathbb{E}[\|\bar{\boldsymbol{g}}\|^2] + \frac{1}{B}\mathbb{E}[\|\boldsymbol{g}_i\|^2]}\left(\|\bar{\boldsymbol{g}}\|^2 + \mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2] + \frac{(B-1)^2\|\bar{\boldsymbol{g}}\|^2}{\|\bar{\boldsymbol{g}}\|^2 + \mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2]}\left(\|\bar{\boldsymbol{g}}\|^2 + \frac{\mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2]}{C}\right)\right.\\ &\qquad \left. +2(B-1)\|\bar{\boldsymbol{g}}\|^2 + \frac{B-1}{C}\mathbb{E}[\|\tilde{\boldsymbol{g}}_i\|^2]\right) \qquad (43)\end{aligned}$$

Combining this with the previous expression for $\mathbb{E}[y_b^2]$ and the definition (D2) of the signal-to-noise ratio $\varphi := \mathbb{E}[\|\bar{\boldsymbol{g}}\|^2]/\mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2]$ we obtain the expression in the appx. E:

$$\begin{aligned} \frac{\mathbb{E}[(\Delta y_b)^2]}{\mathbb{E}[y_b^2]} &= \frac{\eta^2 C}{B^2\|\boldsymbol{w}\|^2}\frac{1}{\mathbb{E}[\|\boldsymbol{g}\|^2]}\left(\mathbb{E}[\|\boldsymbol{g}_b\|^2] + \frac{B-1}{C}\mathbb{E}[\|\tilde{\boldsymbol{g}}_i\|^2]\right.\\ &\qquad \left. + \frac{(B-1)^2}{\mathbb{E}[\|\boldsymbol{g}_b\|^2]}\left(\|\bar{\boldsymbol{g}}\|^4 + \frac{\|\bar{\boldsymbol{g}}\|^2\mathbb{E}[\|\tilde{\boldsymbol{g}}_b\|^2]}{C}\right) + 2(B-1)\|\bar{\boldsymbol{g}}\|^2\right) \qquad (44)\\ &= \frac{\eta^2 C}{B^2\|\boldsymbol{w}\|^2}\frac{1}{\varphi + \frac{1}{B}}\left((\varphi+1) + \frac{B-1}{C} + \left(\frac{(B-1)^2\varphi}{\varphi+1}\left(\varphi + \frac{1}{C}\right) + 2(B-1)\varphi\right)\right) \qquad (45)\end{aligned}$$

G.5. Estimating the Signal-to-Noise Ratio

We use accumulation over the microbatches to estimate the SNR at a given time. Let's assume we have A microbatches of size M each, with the average gradient of a microbatch denoted $\boldsymbol{g}_m$ and the average gradient of the whole batch denoted $\boldsymbol{g} = \frac{1}{A}\sum_m \boldsymbol{g}_m$.

We estimate the variance of the norm of a single gradient example, i.e. the noise power as:

$$P_N = \frac{A}{A-1}\cdot M \cdot \mathbf{1}^\top\left(\frac{1}{A}\sum_m \boldsymbol{g}_m^2 - \boldsymbol{g}^2\right) \tag{46}$$

The signal power is estimated as:

$$P_S = \mathbf{1}^\top\boldsymbol{g}^2 - \frac{1}{AM}P_N \tag{47}$$

Our SNR estimate is then:

$$\varphi = P_S/P_N \tag{48}$$

G.6. RRC Correction Factor

The RRC correction is done based on eq. (11) and the SNR estimation eq. (48). We assume the learning rate was originally scaled with the square root of the batch size, which is derived for an SNR of zero, and downscale the step size to compensate for the measured SNR and batch size. We define:

$$\rho = \frac{1}{B(1+\varphi)}\left((\varphi+1) + \frac{B-1}{C} + \left(\frac{(B-1)^2\varphi}{\varphi+1}\left(\varphi + \frac{1}{C}\right) + 2(B-1)\varphi\right)\right) \tag{49}$$

For numerical purposes, we clamp $1 \leq \rho \leq B$ which corresponds to $\varphi = 0$ and $\varphi = \infty$ for a large $C \rightarrow \infty$. The update scaling factor is the square root of an EMA of the inverse of this quantity. We use the same coefficient as for the momentum and compute this for the matrix of each linear layer independently. This form for the scaling factor is somewhat arbitrary, complicated by the fact that Lion-like algorithms fix the step size exactly, so scaling the gradient at each step size can not change the magnitude of the update. For Adam or SGD like algorithms we could scale the gradient contributions directly instead of scaling the update size.

G.7. Run-to-run Variance / Uncertainty Estimation

We do not quantify the uncertainty for every GPT2 configuration in our sweeps. This would require significantly more compute and our estimates of the uncertainty for select points indicate that this would not qualitatively change our results. For the baseline AdamW run the run-to-run differences in the validation loss over different seeds are around 0.05. However, the relative ranking of different runs remained the same.

G.8. Computational Requirements

Our experiments are performed on A100 GPUs with either 40GB or 80GB of RAM. One training run for our GPT2 setup takes around 4h, running on a single GPU. Reproducing the GPT2 experiments reported should take on the order of 1000 GPU hours. Including our preliminary experiments brings this up to around 3x this amount.

Appendix H. Limitations

Our main experiments focus on a single network which may not be broad enough to generalize to a wide range of networks. We believe we identify real factors that contribute to the need for warmup, but these may not be the only ones across a broader range of networks. Similarly, the promising results for reducing or eliminating the warmup with higher momentum values or the relative representation correction would benefit from further validation across additional settings.