Batch size invariant Adam.

Xi Wang XWANG3@CS.UMASS.EDU
University of Massachusetts Amherst
Laurence Aitchison LAURENCE.AITCHISON@BRISTOL.AC.UK
University of Bristol

Abstract

We propose a batch size invariant version of AdamW, which gives almost the same learning trajectories irrespective of the minibatch size. Our batch size invariant AdamW splits the MINI-batch into MICRO-batches. In the large-scale setting, MICRO-batches may be distributed among worker nodes, but MICRO-batch gradients can also be evaluated in parallel on a single node (e.g. using a vmap implementation from Jax or PyTorch). For the v term, standard AdamW first computes the average over MICRO-batch gradients, then squares, while in the batch size invariant AdamW proposed here, we first square the MICRO-batch gradients, then average. Previous work (e.g. Malladi et al. 2022) used an alternative approach that involved a square-root scaling of the learning rate, but this approach only works in the setting where the gradient variance dominates the square of the expected gradient. In contrast, the approach proposed here gives batch size invariance without this assumption. We confirm that in practice our scheme gives batch size invariance in a much larger range of scenarios than the previous approach.

1. Introduction

Adam [10] and AdamW [13] are state-of-art optimizers, commonly used for LLM pretraining runs [e.g. 22, 25]. Given the huge investments in these pretraining runs, it is critical to understand how to tune the hyperparameters of AdamW[1]. However, as with most optimization algorithms, setting the AdamW hyperparameters is difficult. As such, we may want to tune the hyperparameters in smaller-scale settings and transfer those hyperparameters to a single, larger training run at the edge of our compute budget. However, to do that effectively, we need our optimization to behave in the same way for any MINI-batch size. This property, which we call batch size invariance, can be achieved straightforwardly in SGD, by scaling the learning rate linearly with the batch size (i.e. $\eta \propto B$) [4, 5, 11, 14, 16, 18–21, 24]. However, for modern adaptive optimizers such as AdamW [10, 13], it is more difficult to achieve batch size invariance.

To understand the key barrier to obtaining truly batch size invariant AdamW, we need to look at the AdamW update rule. Here, we ignore decay terms as they are not relevant for this explanation,

$$\Delta w_t = -\eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{1}$$

The key term here is $\hat{v}_t$, which is a debiased, exponential moving average estimate of the raw second moment of the gradient, $\mathrm{E}\left[(g')^2\right]$ (here, the expectation is over randomness in the choice of

1. We will use AdamW to denote both Adam and AdamW for the rest of the manuscript, as Adam can be obtained from AdamW by setting the AdamW weight-decay hyperparameter to zero, $\lambda = 0$. At the same time, any Adam weight decay can be incorporated into the objective.

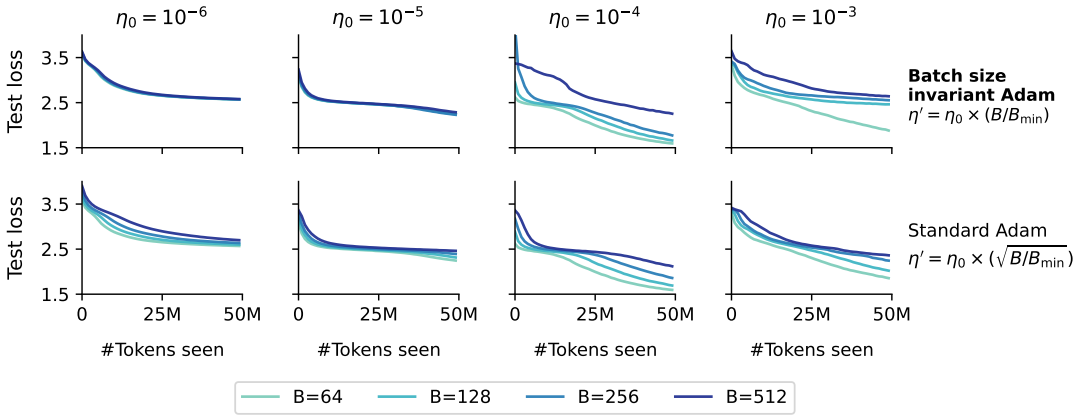

Figure 1: Batch size invariant AdamW (top) v.s. standard AdamW (bottom) on NanoGPT trained on Shakespeare under various batch sizes (B) and base learning rates (η_0) with base batch size $B_{\text{min}} = 64$. Batch size invariant AdamW uses actual learning rate $\eta' \propto B$ while standard AdamW uses $\eta' \propto \sqrt{B}$ [6, 8, 15]. Batch size invariant AdamW shows consistent test loss traces across batch sizes until $\eta_0 = 10^{-4}$ whereas standard AdamW shows discrepancy at the smallest η_0 considered.

datapoints incorporated in the MINI-batch, and we use g' for the MINI-batch gradient as we will use g for the MICRO-batch gradient later in the manuscript). Of course, $\mathrm{E}\left[(g')^2\right] = \mathrm{E}\left[g'\right]^2 + \mathrm{Var}\left[g'\right]$, i.e. the raw second moment is the sum of the mean-squared gradient, $\mathrm{E}\left[g'\right]^2$, and the variance of the gradient, $\mathrm{Var}\left[g'\right]$. The key issue is that the variance of the MINI-batch gradient depends on the MINI-batch size, $\mathrm{Var}\left[g'\right] \propto 1/B$ [6, 8, 15]. There is therefore an inevitable MINI-batch size dependence in $\hat{v}_t$, which appears as a change in the effective learning rate, $\eta/(\sqrt{\hat{v}_t} + \epsilon)$.

Recent work suggested correcting for these effects by tweaking the learning rate, η [6, 8, 15]. In particular, they proposed a square-root scaling (i.e. $\eta \propto \sqrt{B}$). However, for this square-root scaling to be valid, strong assumptions need to hold, namely that $\mathrm{Var}\left[g'\right]$ "dominates" (i.e. is much bigger than) $\mathrm{E}\left[g'\right]^2$. Of course, this can hardly be guaranteed to hold. In fact, there are two settings where this assumption might break down. First, close to initialization, where $\mathrm{E}\left[g'\right]$ will be large. Second, as the batch size gets very large (e.g. in large LLM pretraining runs), we would expect $\mathrm{Var}\left[g'\right] \propto 1/B$ to become small.

We take a different approach to batch size invariance in AdamW, by modifying the AdamW updates themselves to eliminate the batch size dependence at source. In particular, we consider setting in which a MINI-batch is split into a number of MICRO-batches. These MICRO-batches may be evaluated in parallel on a single node, e.g. using a `vmap` implementation from Jax or PyTorch. Alternatively, in the large-scale setting, the MICRO-batches may be distributed among worker nodes. Standard AdamW computes the average gradient (across MICRO-batches), then squares. In contrast, we consider an alternative scheme, which first squares the MICRO-batch gradients, then averages across MICRO-batches. We prove that this alternative scheme is batch size invariant under much weaker assumptions (Appendix. C). In particular, we do not need the gradient variance to dominate the expectation. We just need sufficiently small updates (e.g. learning rates), which is required by Malladi et al. [15] anyway, and is just the AdamW analogue of the critical batch size in SGD [4, 14, 16, 18, 24].

Symbol	Description
κ	Number of MICRO-batches in a MINI-batch.
w_t	Parameter values after consuming t MICRO-batch.
$g_t(w)$	Gradient for the tth MICRO-batch evaluated at w.
η	Learning rate.
(γ_1, γ_2)	EMA parameters (related to β_1 and β_2, Eq. 2).
λ	Weight decay coefficient.

Algorithm 1 MICRO AdamW (i.e. standard AdamW applied to MICRO-batches)

while not converged **do**

$$m_t = (1-\gamma_1)m_{t-1} + \gamma_1 g_t(w_{t-1})$$
$$v_t = (1-\gamma_2)\ v_{t-1} + \gamma_2 g_t^2(w_{t-1})$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1(\gamma_1)^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2(\gamma_2)^t}$$
$$w_t = (1-\eta\lambda)w_{t-1} - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$

Algorithm 2 Standard AdamW applied to MINI-batches.

while not converged **do**

$$m_t = (1-\gamma_1')m_{t-\kappa} + \gamma_1'\frac{1}{\kappa}\sum_{k=1}^{\kappa} g_{t-k+\kappa}(w_{t-\kappa})$$
$$v_t = (1-\gamma_2')\ v_{t-\kappa} + \gamma_2'(\frac{1}{\kappa}\sum_{k=1}^{\kappa} g_{t-k+\kappa}(w_{t-\kappa}))^2$$
$$\hat{m}_t = \frac{m_t'}{1-\beta_1(\gamma_1')^{t/\kappa}}$$
$$\hat{v}_t = \frac{v_t'}{1-\beta_2(\gamma_2')^{t/\kappa}}$$
$$w_t = (1-\eta'\lambda)w_{t-\kappa} - \eta'\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$

Algorithm 3 Batch size invariant AdamW applied to MINI-batches (**ours**).

while not converged **do**

$$m_t = (1-\gamma_1')m_{t-\kappa} + \gamma_1'\frac{1}{\kappa}\sum_{k=1}^{\kappa} g_{t-k+\kappa}(w_{t-\kappa})$$
$$v_t = (1-\gamma_2')\ v_{t-\kappa} + \gamma_2'\frac{1}{\kappa}\sum_{k=1}^{\kappa} g_{t-k+\kappa}^2(w_{t-\kappa})$$
$$\hat{m}_t = \frac{m_t'}{1-\beta_1(\gamma_1')^{t/\kappa}}$$
$$\hat{v}_t = \frac{v_t'}{1-\beta_2(\gamma_2')^{t/\kappa}}$$
$$w_t = (1-\eta'\lambda)w_{t-\kappa} - \eta'\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$

2. Methods

To set the context, we recall the concept of a MICRO-batch. A MICRO-batch is formed by taking a MINI-batch of B datapoints, and splitting it equally into κ MICRO-batches, each of size B/κ. Then we compute the gradients for each of the MICRO-batches. Finally, we do a parameter update, by aggregating the MICRO-batch gradients. Of course, MICRO-batches are widely adopted in the training of deep networks, e.g. MICRO-batching facilitates larger MINI-batch sizes under constrained memory by enabling gradient accumulation and supports data parallelism through distributing MICRO-batches across workers for parallel gradient computation. That said, we can also easily run multiple MICRO-batches in parallel on a single node, e.g. using a `vmap` implementation from Jax or PyTorch. Importantly, our focus diverges from these applications; we leverage MICRO-batching to explore how optimization algorithms are influenced by MINI-batch size.

We began by writing down standard AdamW updates applied to individual MICRO-batches (denoted t) (Alg. 1). Here, $g_t(w)$ is the gradient of the objective for the tth MICRO-batch. Note that

we have written AdamW here in a slightly unusual parameterization, using,

$$\beta_1(\gamma_1) = 1 - \gamma_1 \qquad \beta_2(\gamma_2) = 1 - \gamma_2 \tag{2}$$

as the scalings we derive later on are much more simply expressed in terms of γ_1 and γ_2 than in terms of β_1 and β_2. Note that we can recover Adam from AdamW by setting $\lambda = 0$, and incorporating any Adam weight decay into the objective.

Next, we wrote down the algorithm for standard AdamW updates applied to MINI-batches (Alg. 2). Here, the MINI-batch gradient for a single step are formed by averaging across the MICRO-batch gradients, for all MICRO-batch in the MINI-batch. Importantly, note that we use t to index MICRO-batch, not MINI-batch. That means that updates advance the state in "jumps" of κ steps (e.g. from $w_{t-\kappa}$ to w_t to $w_{t+\kappa}$) because of course, each update operates on a MINI-batch and a MINI-batch is formed of κ MICRO-batches. This unusual convention has the key benefit that the time indices in standard AdamW remains aligned with the time indices in MICRO AdamW (Alg. 1), which will be important as we will be assessing equivalence between MICRO AdamW and standard AdamW or batch size invariant AdamW. However, this notation does have the disadvantage that it modifies slightly the form of the debiasing equations (see the powers of steps $= t/\kappa$ in the expressions for $\hat{m}$ and $\hat{v}$ in Alg. (2) and Alg. (3)).

Batch size dependence in standard AdamW. A standard AdamW update (Alg. 2) applies to a MINI-*batch* formed by merging κ MICRO-batches. However, the arguments given in the introduction indicate that standard AdamW is not batch size invariant. By "batch size invariant", we mean that, in some sensible limit, we get the same learning trajectories if we fix the MICRO-batch size, but vary the MINI-batch size by taking different numbers of microbatches, κ. To see that standard AdamW (Alg. 2) is not batch size invariant, we consider the $\hat{v}_t$ terms in the various algorithms. For MICRO AdamW (Alg. 1), $\hat{v}_t$ is an exponential moving average of the raw second moment of the MICRO-*batch* gradients,

$$\mathrm{E}\left[g_t^2(w_{t-1})\right] = \mathrm{E}\left[g\right]^2 + \mathrm{Var}\left[g\right]. \tag{3}$$

where $\mathrm{E}\left[g\right]$ and $\mathrm{Var}\left[g\right]$ is the mean and variance of MICRO-batch gradients. In contrast, for standard AdamW (Alg. 2), $\hat{v}_t$ is an exponential moving average of the raw second moment of the *minibatch* gradients

$$\mathrm{E}\left[(\tfrac{1}{\kappa}\textstyle\sum_{k=1}^{\kappa} g_{t-k+\kappa}(w_{t-\kappa}))^2\right] = \mathrm{E}\left[g\right]^2 + \tfrac{1}{\kappa}\,\mathrm{Var}\left[g\right]. \tag{4}$$

Critically, as we have more MICRO-batches, κ, in the MINI-batch, we get a better estimate of the gradient and hence the variance term falls. And as the effective learning rate, $\eta/(\sqrt{\hat{v}_t} + \epsilon)$ depends on $\hat{v}_t$, we can see that the effective learning rate depends unavoidably on the batch size.

Batch size invariant AdamW. We therefore consider a batch size invariant version of AdamW (Alg. 3) in which the red term is the average over squared MICRO-batch gradients (as opposed to standard AdamW, where the red term is the square of average MICRO-batch gradients). Thus, in batch size invariant AdamW (Alg. 3), just like in MICRO AdamW (Alg. 1), $\hat{v}$ estimates the expected raw second moment of the fixed-size MICRO-batch gradients,

$$\mathrm{E}\left[\tfrac{1}{\kappa}\textstyle\sum_{k=1}^{\kappa} g_{t-k+\kappa}^2(w_{t-\kappa})\right] = \mathrm{E}\left[g\right]^2 + \mathrm{Var}\left[g\right]. \tag{5}$$

This expectation thus does not depend on κ, avoiding a key source of batch size dependence. Proof for the batch size invariance property is provided in Appendix. C, where we prove that the trajectories for batch size invariant AdamW applied to any MINI-batch size (formed by κ MICRO-batches) are equivalent to the trajectory from κ steps of MICRO AdamW on the same underlying MICRO-batches (Alg. 1).

3. Experiments

In this section, we empirically confirmed that batch size invariant AdamW with sufficiently small learning rates indeed gave almost exactly equivalent optimization results under different batch sizes. We additionally compared its optimization trajectories against standard AdamW under the square root scaling rule, where we observed a higher level of inconsistency across different batch sizes.

Optimization setting We considered two optimization methods: Batch size invariant AdamW with actual learning rate η' scaled linearly with the batch size B:

$$\eta' = \eta_0 B / B_{\min}, \tag{6}$$

and standard AdamW, with η' proportion to $\sqrt{B}$:

$$\eta' = \eta_0 \sqrt{B / B_{\min}}. \tag{7}$$

where η_0 is the base learning rate and $B_{\min}$ is the smallest batch size used for a particular task. For both methods, we set the running average parameters [8] as

$$\gamma_1' = 0.1 \times \frac{B}{B_{\min}} \qquad \gamma_2' = 0.001 \times \frac{B}{B_{\min}}. \tag{8}$$

For all tasks and both optimizers, we swiped η_0 in range $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ and used $B_{\min}$ as the MICRO-batches for batch size invariant AdamW.

Results The results are presented in Fig. 1 (NanoGPT on Shakespeare), Fig. 2, 3, 4 (ResNet-18 with batchnorm, ResNet-18 with layernorm, and ViT on CIFAR-10). Full model descriptions are presented in Appendix. B. All figures show the model's performance metrics trace for different learning rates and batch sizes averaged over three random trials. Broadly, we observed that under our proposed batch size invariant AdamW, the trajectories under different batch sizes almost exactly overlap for small learning rates (note that expect batch size invariance breaking down at some point as the learning rate increases is expected). In contrast, the standard AdamW with square-root scaling (Fig. 2 left) reveals more pronounced differences across batch sizes across all learning rates considered, including η_0 as small as 10^{-7}.

4. Limitation

It is worth pointing out that, batch size invariant AdamW still demonstrates discrepancies between batch sizes after the learning rate exceeds some certain "critical learning rate", and this critical learning rate (e.g. 10^{-4} for experiments in Fig. 2) is far smaller than the commonly used value and results in slow convergence. We believe such discrepancy is inevitable in that the batch size invariant guarantee, similar to [15], is based on the assumption that the learning rate is sufficiently small, which is an unknown amount that may vary between models and tasks.

References

[1] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

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

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[4] Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*, 2018.

[5] 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.

[6] Diego Granziol, Stefan Zohren, and Stephen Roberts. Learning rates as a function of batch size: A random matrix theory approach to neural network training. *The Journal of Machine Learning Research*, 23(1):7795–7859, 2022.

[7] 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.

[8] Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization. *Advances in Neural Information Processing Systems*, 35:17086–17098, 2022.

[9] Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

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

[11] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[14] Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, pages 3325–3334. PMLR, 2018.

[15] Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. On the sdes and scaling rules for adaptive gradient algorithms. *Advances in Neural Information Processing Systems*, 35: 7697–7711, 2022.

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

[17] Fabian Schaipp, Ruben Ohana, Michael Eickenberg, Aaron Defazio, and Robert M Gower. Momo: Momentum models for adaptive learning rates. *arXiv preprint arXiv:2305.07583*, 2023.

[18] 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.

[19] Samuel L Smith and Quoc V Le. A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*, 2017.

[20] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[21] Mandt Stephan, Matthew D Hoffman, David M Blei, et al. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 18(134):1–35, 2017.

[22] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[23] 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.

[24] 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.

[25] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Appendix A. Related work

The closest prior work is Granziol et al. [6], Malladi et al. [15] and Hilton et al. [8], as they propose an alternative approach to obtaining batch size invariant AdamW by scaling the learning rate to try to post-hoc correct for changes in the effective learning rate, $\eta/(\sqrt{\hat{v}}+\epsilon)$, caused by changes in $\hat{v}$. In contrast, we propose to eliminate changes in the effective learning rate at source, by proposing a modification to AdamW updates. There are three key differences between our approach and this prior work. First, our approach gives a linear scaling of the learning rate, $\eta \propto B$, while the prior work proposes a square-root scaling, $\eta \propto \sqrt{B}$ (this is not a contradiction: both are correct in their respective setups). Second, our theoretical approaches are radically different. In particular, Malladi et al. [15] use stochastic differential equations. In particular, they take the choice of datapoints in the MINI-batch to be random, and they consider the updates to be random variables. In contrast, in our actual proof (Appendix F), we consider the choice of datapoints in the MINI-batches as fixed, and thus the updates become deterministic. Instead, we ask: if we give the same set of datapoints to two optimizers, in what settings are the resulting weight updates exactly equivalent? Third, our approach gives batch size invariance under much weaker assumptions. In particular, we do not need the gradient variance to dominate the expectation, we only need the updates to be sufficiently small (which is required by Malladi et al. [15] anyway).

It turns out that this requirement for the updates to be sufficiently small is also encountered in SGD, where it is understood as a critical batch size [14]. Notably, this work assumes takes $\eta \propto B$ so a critical batch size is intimately related to a critical learning rate. Shallue et al. [18] introduced the term "perfect" scaling for the region in which training is batch size invariant and studied the effect of e.g. architectures on the critical batch size. The dependence of the critical batch size on dataset [4], gradient noise scale [16] and curvature Zhang et al. [24]. We observe a similar threshold for batch size invariant AdamW (Fig. 3), though in our plots it is most straightforwardly interpreted as a critical learning rate.

Appendix B. Full experiment settings and results

Task description We trained ResNet-18 [7], vision transformer [3, ViT] on CIFAR-10 [12] and NanoGPT [9] on Shakespeare dataset. Note that, batchnorm inside ResNet-18 itself can introduce batch size dependent effects (which arise because larger batches imply more accurate estimates of the feature means and variances). Of course this does not affect our batch size invariant AdamW, as those batchnorm means and variances are computed for the MICRO-batches. But it is important to check that the batch size effects observed using Malladi et al. [15]'s square-root scaling are not just an artifact of batchnorm. We therefore took the ResNet-18 (Fig. 2) and replaced batchnorm with layernorm. We still found that our batch size invariant AdamW offered considerable improvements in consistency across batch sizes (Fig. 3).

For ResNet-18, we used standard cross-entropy loss with random cropping and flipping as data augmentation. For ViT[2], we used a patch size of 8 and we added QK-layernorm [2, 23] for more stabilized training under larger learning rates. We additionally incorporated label smoothing loss and automatic data augmentation [1] to match up the test performance with the numbers reported in other literature, e.g. Schaipp et al. [17]. For NanoGPT, we followed the model configuration in code

2. Implementation based on https://github.com/kuangliu/pytorch-cifar/

base [3], which used a decoder-only transformer with 6 attention layers, and 6 attention heads with an embedding dimension of 384.

We used constant learning with a weight decay of $\lambda = 0.01$ for ResNet and ViT experiments. For NanoGPT, we used cosine decay learning rate decay to a $1/10$ of the initial learning rate and a weight decay of 0.1, following the original codebase's setting.

Appendix C. Theorem statements

While we have clearly eliminated one source of batch size dependence, we have yet to prove that Alg. (3) really is batch size invariant in any formal sense. We begin with a theorem that states that with an appropriate choice of hyperparameters, κ steps of MICRO AdamW on MICRO-batches of size M gives equivalent optimizer-state-updates (i.e. updates for m_t, v_t and g_t) to a single step of batch size invariant AdamW with the same datapoints grouped into a single MINI-batch of size $B = \kappa M$ (Theorem 1). This implies that with an appropriate choice of hyperparameters, batch size invariant AdamW with different MINI-batch sizes gives equivalent optimizer-state-updates after consuming the same set of data points (Theorem 2). For the proofs, see Appendices D–G.

To understand the formal statement of the theorem, we need to understand how to formally obtain small η, γ_1 and γ_2. We do this by setting,

$$\eta = \delta\bar{\eta}_0 \tag{9a}$$
$$\gamma_1 = \delta\bar{\gamma}_1 \tag{9b}$$
$$\gamma_2 = \delta\bar{\gamma}_2. \tag{9c}$$

and taking $\delta \to 0$. Of course, as $\delta \to 0$, all state variables (w_t, m_t and v_t) stop changing so the updates from the two optimizers are trivially equivalent. To avoid this trivial equivalence, we actually consider the equivalence of weight changes, divided by δ (see below).

Theorem 1 *Consider two optimizers:* MICRO *AdamW (Alg. 1; i.e. standard AdamW applied to* MICRO*-batches) with hyperparameters η, γ_1 and γ_2, and batch size invariant AdamW (Alg. 3) with hyperparameters*

$$\eta' = \kappa\eta, \tag{10a}$$
$$\gamma_1' = \kappa\gamma_1, \tag{10b}$$
$$\gamma_2' = \kappa\gamma_2. \tag{10c}$$

applied to MINI*-batches composed of κ* MICRO*-batches. We start both optimizers at time $t - \kappa$ at the same initial state, $w_{t-\kappa}$, $m_{t-\kappa}$ and $v_{t-\kappa}$. We take w_t to be the result of applying κ steps of standard AdamW to κ* MICRO*-batches, and w_t' to be the result of applying a single step of batch size invariant AdamW to a* MINI*-batch (consisting of the same κ* MICRO*-batches merged together). Then,*

$$\lim_{\delta\to 0} \frac{m_t - m_{t-\kappa}}{\delta} = \lim_{\delta\to 0} \frac{m_t' - m_{t-\kappa}}{\delta} \tag{11a}$$
$$\lim_{\delta\to 0} \frac{v_t - v_{t-\kappa}}{\delta} = \lim_{\delta\to 0} \frac{v_t' - v_{t-\kappa}}{\delta} \tag{11b}$$
$$\lim_{\delta\to 0} \frac{w_t - w_{t-\kappa}}{\delta} = \lim_{\delta\to 0} \frac{w_t' - w_{t-\kappa}}{\delta}. \tag{11c}$$

3. https://github.com/karpathy/nanoGPT/blob/master/config/train_shakespeare_char.py

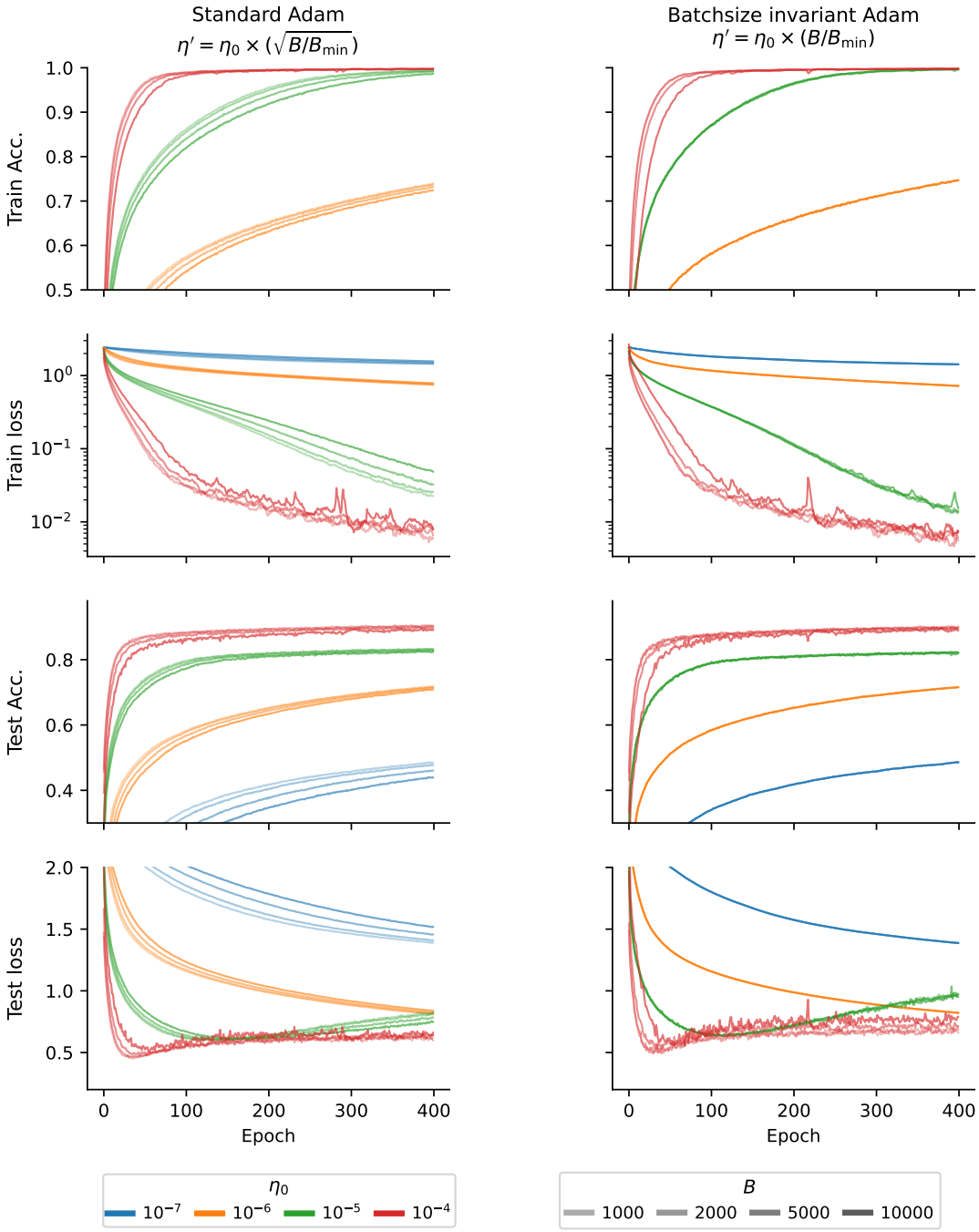

Figure 2: Comparing the behavior of our proposed batch size invariant AdamW (right column), with $\eta \propto B$ against standard AdamW (left column), with $\eta \propto \sqrt{B}$ [6, 8, 15]. The model was a ResNet-18 trained on CIFAR-10 over 200 epochs, under batch sizes (opacity) ranging from $B = 1000$ to $B = 10000$, with $B_{\min} = 1000$. Each color represents a different base learning rate η_0. Note that batch size invariant AdamW (right) gives almost perfect batch size invariance (in that the lines are all on top of each other) up until $\eta = 10^{-5} \times (B/B_{\min})$. In contrast, with standard AdamW (left), you get discrepancies even with the smallest learning rate, i.e. $\eta_0 = 10^{-7}$.

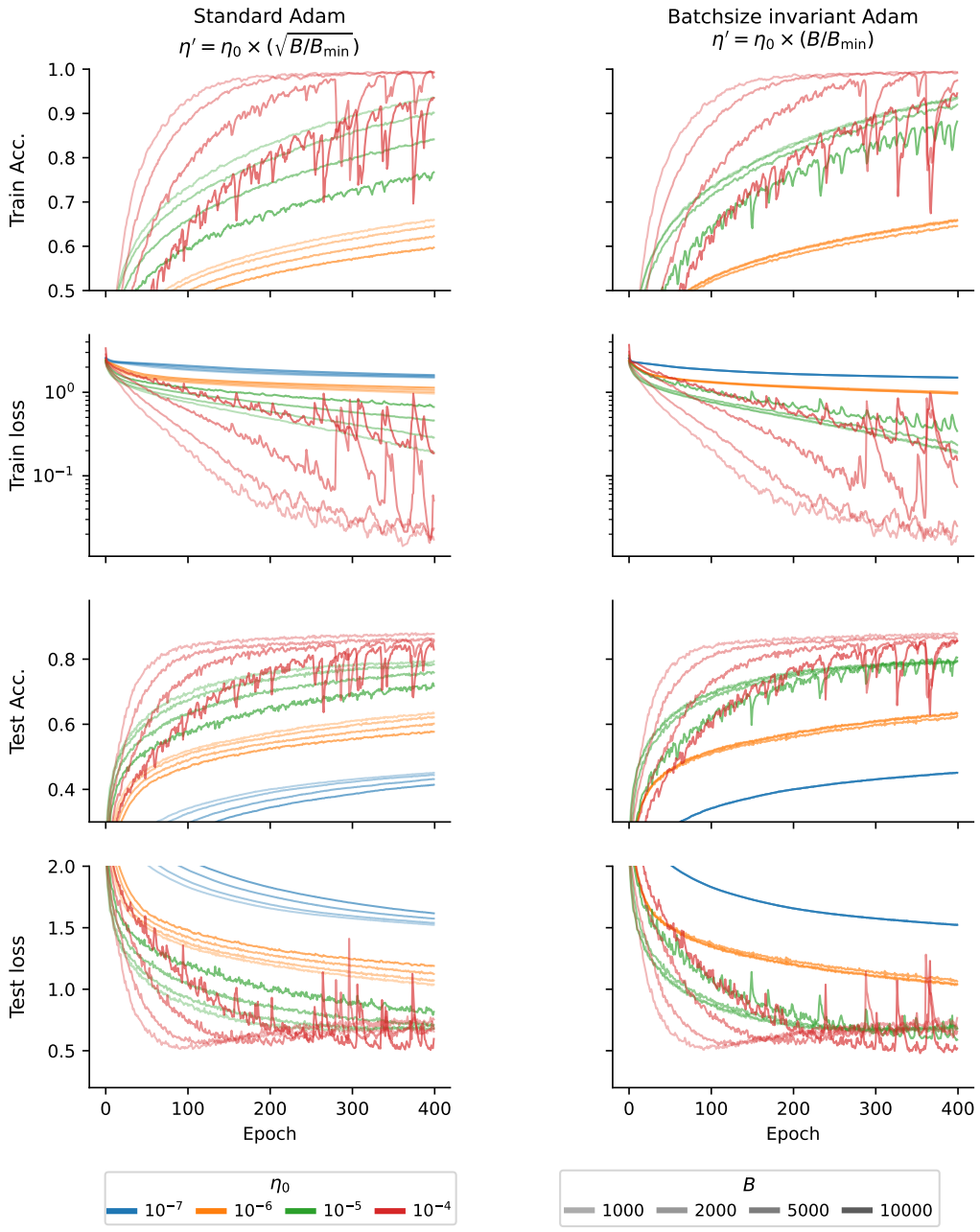

Figure 3: As Fig. 2, but with layernorm rather than batchnorm. In particular, we compare the behavior of our proposed batch size invariant AdamW, with $\eta \propto B$ (right) against standard AdamW, with $\eta \propto \sqrt{B}$ (left), both with $B_{\min} = 1000$. Similar to the batchnorm results, the batch size invariant AdamW lines (right) almost exactly line up, until $\eta = 10^{-4} \times (B/B_{\min})$. Whereas standard AdamW (left) shows discrepancies between lines even under the smallest learning rate considered ($\eta_0 = 10^{-7}$).

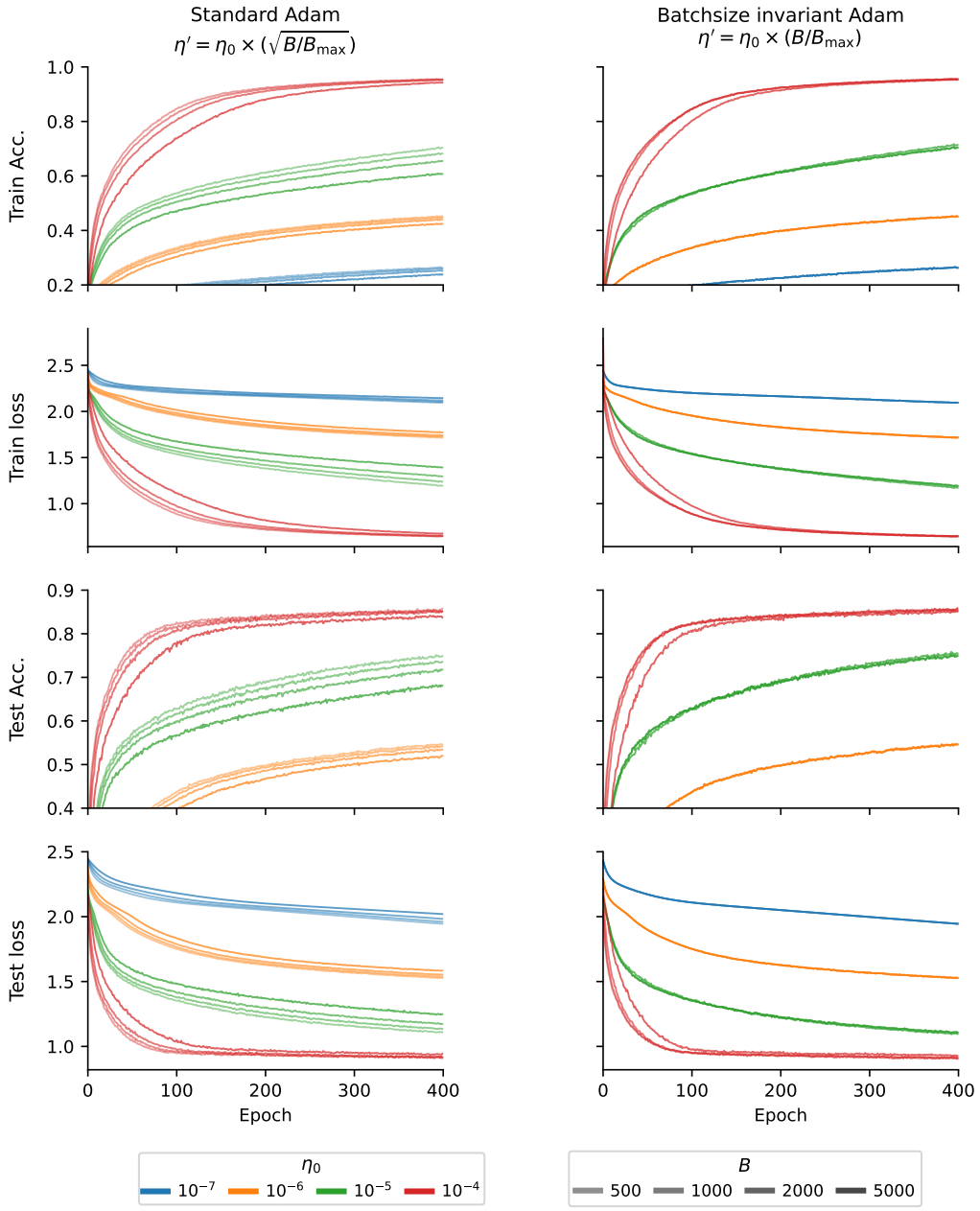

Figure 4: As Fig. 2, but use ViT rather than ResNet-18. Again, we compare the behavior of our proposed batch size invariant AdamW, with $\eta \propto B$ (right) against standard AdamW, with $\eta \propto \sqrt{B}$ (left). A $B_{\min}$ of 500 is used. Standard AdamW shows aligned trajectories for the smallest and largest base learning rate but shows discrepancy at all values of η_0. This is expected, as obtaining batch size invariance with standard AdamW under the square-root scaling requires the gradient variance dominating the gradient mean, which may not hold when the parameters are near initialization, where the gradient may have a large magnitude. Regardless, our proposed batchsize invariant AdamW shows consistency across all learning rates considered at all stages in the optimization.

i.e. the state updates for the merged and unmerged optimizers are equivalent for sufficiently small η, γ_1 and γ_2.

Theorem 2 *Consider batch size invariant AdamW under two* MINI*-batch sizes $B' = \kappa' M$ and $B'' = \kappa'' M$, where M denotes the size of the* MICRO*-batch, with hyperparameters*

$$\eta' = \kappa' \eta_0, \qquad \eta'' = \kappa'' \eta_0 \tag{12a}$$
$$\gamma_1' = \kappa' \gamma_1, \qquad \gamma_1'' = \kappa'' \gamma_1 \tag{12b}$$
$$\gamma_2' = \kappa' \gamma_2, \qquad \gamma_2'' = \kappa'' \gamma_2 \tag{12c}$$

Consider an integer number of MICRO*-batches, K, which can be divided by both κ' and κ''. We start both optimizers at time $t - K$ with state $m_{t-K}, v_{t-K}, w_{t-K}$. We take K/κ' update steps with the first optimizer, which results in an optimizer state of m_t', v_t', w_t'. We also take K/κ'' update steps with the second optimizer, which results in an optimizer state of m_t'', v_t'', w_t''. Then,*

$$\lim_{\delta \to 0} \frac{m_t' - m_{t-K}}{\delta} = \lim_{\delta \to 0} \frac{m_t'' - m_{t-K}}{\delta} \tag{13a}$$
$$\lim_{\delta \to 0} \frac{v_t' - v_{t-K}}{\delta} = \lim_{\delta \to 0} \frac{v_t'' - v_{t-K}}{\delta} \tag{13b}$$
$$\lim_{\delta \to 0} \frac{w_t' - w_{t-K}}{\delta} = \lim_{\delta \to 0} \frac{w_t'' - w_{t-K}}{\delta}. \tag{13c}$$

i.e. the state updates are equivalent for sufficiently small η, γ_1 and γ_2.

Appendix D. Connecting limits in Theorem 1 with gradients

Notice that the quantities in Eq. (11) can be understood as derivatives,

$$\frac{dm_t}{d\delta} = \lim_{\delta \to 0} \frac{m_t - m_{t-\kappa}}{\delta} \qquad \frac{dm_t'}{d\delta} = \lim_{\delta \to 0} \frac{m_t' - m_{t-\kappa}'}{\delta}, \tag{14a}$$
$$\frac{dv_t}{d\delta} = \lim_{\delta \to 0} \frac{v_t - v_{t-\kappa}}{\delta} \qquad \frac{dv_t'}{d\delta} = \lim_{\delta \to 0} \frac{v_t' - v_{t-\kappa}'}{\delta}, \tag{14b}$$
$$\frac{dw_t}{d\delta} = \lim_{\delta \to 0} \frac{w_t - w_{t-\kappa}}{\delta} \qquad \frac{dw_t'}{d\delta} = \lim_{\delta \to 0} \frac{w_t' - w_{t-\kappa}'}{\delta}. \tag{14c}$$

To obtain this result, we used the observation that for $\delta = 0$, the state does not change, so

$$w_t(\delta = 0) = w_{t-\kappa} \qquad w_t'(\delta = 0) = w_{t-\kappa}, \tag{15a}$$
$$m_t(\delta = 0) = m_{t-\kappa} \qquad m_t'(\delta = 0) = m_{t-\kappa}, \tag{15b}$$
$$v_t(\delta = 0) = v_{t-\kappa} \qquad v_t'(\delta = 0) = v_{t-\kappa}. \tag{15c}$$

Thus, all we need to do is show equivalence of the gradient of the state (m_t, v_t and w_t) wrt δ.

Appendix E. Warmup by proving a similar result for SGD

To prove the equivalance of the merged and unmerged optimizers, it is easiest to first consider SGD. A single SGD step can be written,

$$g_t = \mathrm{grad}_t(w_{t-1}) \tag{16a}$$
$$w_t = \mathrm{sgd}(w_{t-1}, g_t, \eta) \tag{16b}$$

where grad_t is a function that computes the gradients for the tth microbatch, and sgd is a function that applies a single SGD update,

$$\mathrm{grad}_t(w) = \nabla \mathcal{L}_t(w_{t-1}), \tag{17a}$$
$$\mathrm{sgd}(w, g, \eta) = (1 - \lambda\eta)w - \eta g. \tag{17b}$$

We have written the updates in this slightly unusual form to make it possible to carefully distinguish partial and total derivatives, which turns out to be important in our setting to apply the chain rule correctly. Specifically, the chain rule applied to g_t and w_t gives,

$$\frac{dg_t}{d\eta} = \frac{\partial g_t}{\partial w_{t-1}}\frac{dw_{t-1}}{d\eta}, \tag{18a}$$
$$\frac{dw_t}{d\eta} = \frac{\partial w_t}{\partial w_{t-1}}\frac{dw_{t-1}}{d\eta} + \frac{\partial w_t}{\partial g_t}\frac{dg_t}{d\eta} + \frac{\partial w_t}{\partial \eta}. \tag{18b}$$

The partial derivatives here are formally defined as,

$$\frac{\partial g_t}{\partial w_{t-1}} = \lim_{h\to 0} \frac{\mathrm{grad}_t(w_{t-1} + h) - \mathrm{grad}_t(w_{t-1})}{h} \tag{19a}$$
$$\frac{\partial w_t}{\partial w_{t-1}} = \lim_{h\to 0} \frac{\mathrm{sgd}(w_{t-1} + h, g_t, \eta) - \mathrm{sgd}(w_{t-1}, g_t, \eta)}{h} \tag{19b}$$
$$\frac{\partial w_t}{\partial g_t} = \lim_{h\to 0} \frac{\mathrm{sgd}(w_{t-1}, g_t + h, \eta) - \mathrm{sgd}(w_{t-1}, g_t, \eta)}{h} \tag{19c}$$
$$\frac{\partial w_t}{\partial \eta} = \lim_{h\to 0} \frac{\mathrm{sgd}(w_{t-1}, g_t, \eta + h) - \mathrm{sgd}(w_{t-1}, g_t, \eta)}{h} \tag{19d}$$

i.e. the partial derivatives compute the change in the output of the *function* (either grad_t or sgd) as one of the arguments to those functions changes, while all the other arguments are held fixed. In contrast, the total derivatives ($d/d\eta$) represent the total change through the whole "compute graph", i.e. the total change in w_t and g_t if we start from a fixed $w_{t-\kappa}$ and change η.

To compute the total derivative for w_t, we use the following partial derivatives,

$$\frac{\partial w_t}{\partial w_{t-1}} = 1 - \lambda\eta \tag{20a}$$
$$\frac{\partial w_t}{\partial g_t} = -\eta \tag{20b}$$
$$\frac{\partial w_t}{\partial \eta} = -g_t - \lambda w_{t-1}. \tag{20c}$$

Substituting these partial derivatives into the total derivative,

$$\frac{dw_t}{d\eta} = (1 - \eta\lambda)\frac{dw_{t-1}}{d\eta} - \eta\frac{dg_t}{d\eta} - g_t - \lambda w_{t-1}. \tag{21}$$

Evaluating at $\eta = 0$,

$$\left.\frac{dw_t}{d\eta}\right|_{\eta=0} = \left.\frac{dw_{t-1}}{d\eta}\right|_{\eta=0} - g_t - \lambda w_{t-1}, \tag{22}$$

we get a simple recursive expression.

Then, if we fix $w_{t-\kappa}$, and consider computing w_t through κ steps of gradient descent, we get,

$$\left.\frac{dw_t}{d\eta}\right|_{\eta=0} = -\sum_{k=1}^{\kappa} \left(g_{t-\kappa+k} + \lambda w_{t-\kappa+k-1}\right), \tag{23}$$

We can simplify this expression by noting that at $\eta = 0$, the weights at all timesteps are equal, $w_{t-1} = w_{t-2} = \cdots = w_{t-k}$. This has important implications for the gradient terms. Specifically, g_t is the gradient for the tth datapoint/microbatch, evaluated using the model parameters at timestep $t-1$, i.e. w_{t-1}. But if the weights do not change, the average of these gradients can be computed based on the initial set of weights (i.e. based on $w_{t-\kappa}$),

$$\left.\tfrac{1}{\kappa}\textstyle\sum_{k=1}^{\kappa} g_{t-k+\kappa}\right|_{\eta=0} = \left.\tfrac{1}{\kappa}\textstyle\sum_{k=1}^{\kappa} \mathrm{grad}_{t-k+\kappa}(w_{t-k+\kappa-1})\right|_{\eta=0} = \tfrac{1}{\kappa}\textstyle\sum_{k=1}^{\kappa} \mathrm{grad}_{t-k+\kappa}(w_{t-\kappa}) = g'_t. \tag{24}$$

Here, g'_t is this average gradient for data across all κ steps, but evaluated using the initial weights, $w_{t-\kappa}$. Thus the gradient resulting from the κ updates is,

$$\left.\frac{dw_t}{d\eta}\right|_{\eta=0} = -\kappa\lambda w_{t-\kappa} - \kappa g'_t. \tag{25}$$

This is starting to resemble a single step of a merged optimizer which performs a single step using a minibatch consisting of the κ minibatches.

However, to be sure of the connection to a single step of a merged optimizer, we need to formally define this optimizer.

$$g'_t = \mathrm{grad}'_t(w_{t-\kappa}) \tag{26a}$$
$$w'_t = \mathrm{sgd}(w_{t-\kappa}, g'_t, \eta') \tag{26b}$$

where grad'_t is a function that computes the gradients for the tth *minibatch*,

$$\mathrm{grad}'_t(w) = \tfrac{1}{\kappa}\textstyle\sum_{k=1}^{\kappa} \mathrm{grad}_{t-\kappa+k}(w) \tag{27}$$

Now, we compute the gradient of w'_t wrt η, taking $\eta' = \kappa\eta$,

$$\left.\frac{dw'_t}{d\eta}\right|_{\eta=0} = -\kappa\lambda w_{t-\kappa} - \kappa g'_t. \tag{28}$$

This is exactly equal to the gradient wrt η of the multi-step gradient in Eq. (25), which establishes our result.

Appendix F. Proof of Theorem 1

In Appendix D, we established that the limits in Theorem 1 can be understood as derivatives (Eq. 14).

However, computing gradients through multiple Adam updates is not trivial. To do so correctly, we need to be careful to distinguish partial and total derivatives. To that end, we write MICRO Adam

updates (Alg. 1) (i.e. standard Adam applied to microbatches) in an unusual form,

$$g_t = \text{grad}_t(w_{t-1}) \tag{29a}$$
$$m_t = \text{ema}_1(m_{t-1}, g_t, \delta) \tag{29b}$$
$$v_t = \text{ema}_2(v_{t-1}, g_t, \delta) \tag{29c}$$
$$u_t = \text{norm\_ema}_t(m_t, v_t) \tag{29d}$$
$$w_t = \text{update}(w_{t-1}, u_t, \delta) \tag{29e}$$

where,

$$\text{ema}_1(m, g, \delta) = (1 - \delta\bar{\gamma}_1)m + \delta\bar{\gamma}_1 g \tag{30a}$$
$$\text{ema}_2(v, g, \delta) = (1 - \delta\bar{\gamma}_2)v + \delta\bar{\gamma}_2 g^2 \tag{30b}$$
$$\text{update}(w, u, \delta) = (1 - \delta\bar{\eta}\lambda)w - \delta\bar{\eta}u. \tag{30c}$$
$$\text{norm\_ema}_t(m, v) = \frac{\hat{m}_t(m)}{\sqrt{\hat{v}_t(v)} + \epsilon} \tag{30d}$$

where,

$$\hat{m}_t(m) = \frac{m}{1 - \beta_1^t} = \frac{m}{1 - (1 - \delta\bar{\gamma}_1)^t} \tag{30e}$$
$$\hat{v}_t(v) = \frac{v}{1 - \beta_2^t} = \frac{v}{1 - (1 - \delta\bar{\gamma}_2)^t} \tag{30f}$$

We are interested in the three state variables, m_t, v_t and w_t, that propagate across timesteps. Using the chain rule, the gradients for these variables can be written,

$$\frac{dm_t}{d\delta} = \frac{\partial m_t}{\partial g_t}\frac{dg_t}{d\delta} + \frac{\partial m_t}{\partial m_{t-1}}\frac{dm_{t-1}}{d\delta} + \frac{\partial m_t}{\partial \delta} \tag{31a}$$
$$\frac{dv_t}{d\delta} = \frac{\partial v_t}{\partial g_t}\frac{dg_t}{d\delta} + \frac{\partial v_t}{\partial v_{t-1}}\frac{dv_{t-1}}{d\delta} + \frac{\partial v_t}{\partial \delta} \tag{31b}$$
$$\frac{dw_t}{d\delta} = \frac{\partial w_t}{\partial w_{t-1}}\frac{dw_{t-1}}{d\delta} + \frac{\partial w_t}{\partial u_t}\frac{du_t}{d\delta} + \frac{\partial w_t}{\partial \delta} \tag{31c}$$

We begin by considering the first state variable, m_t. We substitute the partial derivatives,

$$\frac{\partial m_t}{\partial g_t} = \delta\bar{\gamma}_1 \tag{32a}$$
$$\frac{\partial m_t}{\partial m_{t-1}} = 1 - \delta\bar{\gamma}_1 \tag{32b}$$
$$\frac{\partial m_t}{\partial \delta} = \bar{\gamma}_1(g_t - m_{t-1}), \tag{32c}$$

into Eq. (31a),

$$\frac{dm_t}{d\delta} = \delta\bar{\gamma}_1\frac{dg_t}{d\delta} + (1 - \delta\bar{\gamma}_1)\frac{dm_{t-1}}{d\delta} + \bar{\gamma}_1(g_t - m_{t-1}). \tag{33}$$

Now, we evaluate at $\delta = 0$,

$$\left.\frac{dm_t}{d\delta}\right|_{\delta=0} = \left.\frac{dm_{t-1}}{d\delta}\right|_{\delta=0} + \bar{\gamma}_1(g_t - m_{t-1}), \tag{34}$$

which gives a simple, recursive result.

We follow almost exactly the same procedure for v_t. We substitute the partial derivatives,

$$\frac{\partial v_t}{\partial g_t} = 2\delta\bar{\gamma}_2 g_t, \tag{35a}$$

$$\frac{\partial v_t}{\partial v_{t-1}} = 1 - \delta\bar{\gamma}_2, \tag{35b}$$

$$\frac{\partial v_t}{\partial \delta} = \bar{\gamma}_2(g_t^2 - v_{t-1}). \tag{35c}$$

into Eq. (31b),

$$\frac{dv_t}{d\delta} = 2\delta\bar{\gamma}_2 g_t \frac{dg_t}{d\delta} + (1 - \delta\bar{\gamma}_2)\frac{dv_{t-1}}{d\delta} + \bar{\gamma}_2(g_t^2 - v_{t-1}). \tag{36}$$

Now, we evaluate at $\delta = 0$,

$$\left.\frac{dv_t}{d\delta}\right|_{\delta=0} = \left.\frac{dv_{t-1}}{d\delta}\right|_{\delta=0} + \bar{\gamma}_2(g_t^2 - v_{t-1}), \tag{37}$$

which gives a simple, recursive result.

Finally, we follow a similar process for w_t. We substitute the partial derivatives,

$$\frac{\partial w_t}{\partial u_t} = \delta\bar{\eta} \tag{38a}$$

$$\frac{\partial w_t}{\partial w_{t-1}} = 1 - \delta\bar{\eta}\lambda \tag{38b}$$

$$\frac{\partial w_t}{\partial \delta} = -\bar{\eta}(u_t + \lambda w_{t-1}). \tag{38c}$$

into Eq. (31c),

$$\frac{dw_t}{d\delta} = \delta\bar{\eta}\frac{du_t}{d\delta} + (1 - \delta\bar{\eta}\lambda)\frac{dw_{t-1}}{d\delta} - \bar{\eta}(u_t + \lambda w_{t-1}). \tag{39}$$

Now, we evaluate at $\delta = 0$,

$$\left.\frac{dw_t}{d\delta}\right|_{\delta=0} = \left.\frac{dw_{t-1}}{d\delta}\right|_{\delta=0} - \bar{\eta}(u_t + \lambda w_{t-1}). \tag{40}$$

which gives a simple, recursive result.

Now, we consider fixing $m_{t-\kappa}$, $v_{t-\kappa}$, and $w_{t-\kappa}$ and performing κ steps of the optimizer

$$\left.\frac{dm_t}{d\delta}\right|_{\delta=0} = \bar{\gamma}_1 \sum_{k=1}^{\kappa} (g_{t-\kappa+k} - m_{t-\kappa+k-1}) \tag{41a}$$

$$\left.\frac{dv_t}{d\delta}\right|_{\delta=0} = \bar{\gamma}_2 \sum_{k=1}^{\kappa} \left(g_{t-\kappa+k}^2 - v_{t-\kappa+k-1}\right) \tag{41b}$$

$$\left.\frac{dw_t}{d\delta}\right|_{\delta=0} = -\bar{\eta} \sum_{k=1}^{\kappa} \left(u_{t-\kappa+k} + \lambda w_{t-\kappa+k-1}\right). \tag{41c}$$

Remember that as $\delta \to 0$ all the state variables do not change, and are equal to their initial values,

$$m_{t-\kappa} = m_{t-\kappa+1} = \cdots = m_{t-1} = m_t \tag{42a}$$
$$v_{t-\kappa} = v_{t-\kappa+1} = \cdots = v_{t-1} = v_t \tag{42b}$$
$$w_{t-\kappa} = w_{t-\kappa+1} = \cdots = w_{t-1} = w_t. \tag{42c}$$

Using these constant values, along with Eq. (24),

$$\left.\frac{dm_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\gamma}_1 m_{t-\kappa} + \bar{\gamma}_1 \textstyle\sum_{k=1}^{\kappa} g_{t-\kappa+k} \tag{43a}$$
$$\left.\frac{dv_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\gamma}_2 v_{t-\kappa} + \bar{\gamma}_2 \textstyle\sum_{k=1}^{\kappa} g^2_{t-\kappa+k} \tag{43b}$$
$$\left.\frac{dw_t}{d\delta}\right|_{\delta=0} = -\bar{\eta}(\textstyle\sum_{k=1}^{\kappa} u_{t-\kappa+k} + \kappa\lambda w_{t-\kappa}). \tag{43c}$$

Finally, we consider the $u_{t-\kappa+k}$ terms, which requires us to consider the debiasing steps. Debiasing actually stops doing anything for fixed t, in the limit as $\delta \to 0$, (i.e. $\hat{m}_t(m) = m$). To avoid this trivial equivalence, we therefore consider a limit in which t increases as δ falls,

$$t = \bar{t}/\delta, \tag{44}$$

where $\bar{t}$ is fixed. That makes sense, because δ decreases, all the updates become smaller, so we may want to perform more iterations. We are going to use the standard limit,

$$\lim_{x\to\infty}(1 + \tfrac{a}{x})^{bx} = e^{ab}. \tag{45}$$

In particular, considering the $(1-\bar{\gamma}_1\delta)^t$ term in Eq. (30e), taking $t = \bar{t}/\delta$, $\delta = 1/x$, and identifying $a = -\kappa\bar{\gamma}$ and $b = \bar{t}$,

$$\lim_{\delta\to 0}(1-\bar{\gamma}_1\delta)^t = \lim_{x\to\infty}(1-\tfrac{\bar{\gamma}_1}{x})^{\bar{t}x} = e^{-\bar{\gamma}_1\bar{t}}. \tag{46}$$
$$\lim_{\delta\to 0}(1-\bar{\gamma}_2\delta)^t = \lim_{x\to\infty}(1-\tfrac{\bar{\gamma}_2}{x})^{\bar{t}x} = e^{-\bar{\gamma}_2\bar{t}}. \tag{47}$$

(where the second equation is exactly the same thing, but for v). In this limit, the debiasing steps can be written,

$$\lim_{\delta\to 0}\hat{m}_t(m) = e^{-\bar{\gamma}_1\bar{t}}m, \tag{48}$$
$$\lim_{\delta\to 0}\hat{v}_t(v) = e^{-\bar{\gamma}_2\bar{t}}v. \tag{49}$$

Substituting the result of these debiasing steps into Eq. (43),

$$\left.\frac{dm_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\gamma}_1 m_{t-\kappa} + \bar{\gamma}_1 \textstyle\sum_{k=1}^{\kappa} g_{t-\kappa+k} \tag{50a}$$
$$\left.\frac{dv_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\gamma}_2 v_{t-\kappa} + \bar{\gamma}_2 \textstyle\sum_{k=1}^{\kappa} g^2_{t-\kappa+k} \tag{50b}$$
$$\left.\frac{dw_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\eta}\left(\frac{u_{t-\kappa}e^{-\bar{\gamma}_1\bar{t}}}{\sqrt{v_{t-\kappa}e^{-\bar{\gamma}_2\bar{t}} + \epsilon}} + \lambda w_{t-\kappa}\right). \tag{50c}$$

Overall, these updates resembles a single step of the merged, batch size invariant Adam.

But to be sure, we write batch size invariant Adam updates in Alg. (3) as,

$$g'_t = \text{grad}'_t(w_{t-\kappa}) \tag{51a}$$
$$s'_t = \text{gradsq}'_t(w_{t-\kappa}) \tag{51b}$$
$$m'_t = \text{ema}'_1(m_{t-\kappa}, g'_t, \delta) \tag{51c}$$
$$v'_t = \text{ema}'_2(v_{t-\kappa}, s'_t, \delta) \tag{51d}$$
$$u'_t = \text{norm\_update}'_{t/\kappa}(m'_t, v'_t) \tag{51e}$$
$$w'_t = \text{update}'(w'_{t-1}, u'_t, \delta) \tag{51f}$$

where,

$$\text{grad}'_t(w_{t-\kappa}) = \tfrac{1}{\kappa}\textstyle\sum_{k=1}^{\kappa} \text{grad}_{t-k+\kappa}(w_{t-\kappa}), \tag{52a}$$
$$\text{gradsq}'_t(w_{t-\kappa}) = \tfrac{1}{\kappa}\textstyle\sum_{k=1}^{\kappa} \text{grad}^2_{t-k+\kappa}(w_{t-\kappa}), \tag{52b}$$
$$\text{ema}'_1(m, g, \delta) = (1 - \delta\kappa\bar{\gamma}_1)m + \delta\kappa\bar{\gamma}_1 g \tag{52c}$$
$$\text{ema}'_2(\ v, s, \delta) = (1 - \delta\kappa\bar{\gamma}_2)\ v + \delta\kappa\bar{\gamma}_2 s \tag{52d}$$
$$\text{update}'(w, u, \delta) = (1 - \delta\kappa\bar{\eta}\lambda)w - \delta\kappa\bar{\eta}u. \tag{52e}$$
$$\text{norm\_update}'_{t/\kappa}(m, v) = \frac{\hat{m}'_{t/\kappa}(m)}{\sqrt{\hat{v}'_{t/\kappa}(v)} + \epsilon} \tag{52f}$$

where,

$$\hat{m}'_{t/\kappa}(m) = \frac{m}{1 - (1 - \gamma'_1)^{t/\kappa}} \tag{52g}$$
$$\hat{v}'_{t/\kappa}(v) = \frac{v}{1 - (1 - \gamma'_2)^{t/\kappa}} \tag{52h}$$

Now, the gradient of the state variables for the merged, batch size invariant Adam updates is,

$$\left.\frac{dm'_t}{d\delta}\right|_{\delta=0} = \kappa\bar{\gamma}_1 m_{t-\kappa} + \bar{\gamma}_1 \textstyle\sum_{k=1}^{\kappa} g_{t-k+\kappa} \tag{53a}$$
$$\left.\frac{dv'_t}{d\delta}\right|_{\delta=0} = \kappa\bar{\gamma}_2 v_{t-\kappa} + \bar{\gamma}_2 \textstyle\sum_{k=1}^{\kappa} g^2_{t-k+\kappa} \tag{53b}$$
$$\left.\frac{dw'_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\eta}(u_t + \lambda w_{t-\kappa}). \tag{53c}$$

Comparing these expressions to Eq. (41), we are almost done. It only remains to consider κu_t, again in a limit in which t increases as δ decreases, $t = \bar{t}/\delta$. Again, we consider the $(1 - \gamma'_1)^{t/\kappa}$ term, using the standard limit, $\lim_{x\to\infty}(1 + \frac{a}{x})^{bx} = e^{ab}$, take $\delta = 1/x$, and identify $a = -\bar{\gamma}$ and $b = \bar{t}$,

$$\lim_{\delta\to 0}(1 - \gamma'_1)^{\bar{t}/(\delta\kappa)} = \lim_{\delta\to 0}(1 - \kappa\bar{\gamma}_1\delta)^{\bar{t}/(\delta\kappa)} = \lim_{x\to\infty}(1 - \tfrac{\kappa\bar{\gamma}_1}{x})^{\frac{\bar{t}}{\kappa}x} = e^{-\bar{\gamma}_1\bar{t}}. \tag{54}$$

Thus, the debiasing steps can be written,

$$\lim_{\delta\to 0} \hat{m}'_{t/\kappa}(m) = e^{-\bar{\gamma}_1\bar{t}}m, \tag{55a}$$
$$\lim_{\delta\to 0} \hat{v}'_{t/\kappa}(v) = e^{-\bar{\gamma}_2\bar{t}}v. \tag{55b}$$

Which gives final gradients,

$$\left.\frac{dm'_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\gamma}_1 m_{t-\kappa} + \bar{\gamma}_1 \sum_{k=1}^{\kappa} g_{t-\kappa+k} \tag{56a}$$

$$\left.\frac{dv'_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\gamma}_2 v_{t-\kappa} + \bar{\gamma}_2 \sum_{k=1}^{\kappa} g^2_{t-\kappa+k} \tag{56b}$$

$$\left.\frac{dw'_t}{d\delta}\right|_{\delta=0} = -\kappa\bar{\eta}\left(\frac{u_{t-\kappa}e^{-\bar{\gamma}_1\bar{t}}}{\sqrt{v_{t-\kappa}e^{-\bar{\gamma}_2\bar{t}}+\epsilon}} + \lambda w_{t-\kappa}\right). \tag{56c}$$

These gradients are exactly the same as the gradients for the multi-step updates (Eq. 41), which proves our result.

Appendix G. Proof of Theorem 2

By Theorem 1,

$$\lim_{\delta\to 0}\frac{w'_t - w_{t-K}}{\delta} = \lim_{\delta\to 0}\frac{w_t - w_{t-K}}{\delta} = \lim_{\delta\to 0}\frac{w''_t - w_{t-K}}{\delta} \tag{57a}$$

$$\lim_{\delta\to 0}\frac{m'_t - m_{t-K}}{\delta} = \lim_{\delta\to 0}\frac{m_t - m_{t-K}}{\delta} = \lim_{\delta\to 0}\frac{m''_t - m_{t-K}}{\delta} \tag{57b}$$

$$\lim_{\delta\to 0}\frac{v'_t - v_{t-K}}{\delta} = \lim_{\delta\to 0}\frac{v_t - v_{t-K}}{\delta} = \lim_{\delta\to 0}\frac{v''_t - v_{t-K}}{\delta}. \tag{57c}$$

where m_t, v_t, w_t is the result of applying K steps of Micro-Adam, with hyperparameters η, γ_1, γ_2.